Tree Topology Estimation

by

Rolando Estrada

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Carlo Tomasi, Supervisor

_____
Ronald Parr

_____
Vincent Conitzer

_____
Scott C. Schmidler

_____
Sina Farsiu

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2013

Abstract

Tree Topology Estimation

by

Rolando Estrada

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Carlo Tomasi, Supervisor

_____
Ronald Parr

_____
Vincent Conitzer

_____
Scott C. Schmidler

_____
Sina Farsiu

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2013

Abstract

Tree-like structures are fundamental in nature. A wide variety of two-dimensional imaging techniques allow us to image trees. However, an image of a tree typically includes spurious branch crossings, and the original relationships of ancestry among edges may be lost. We present a methodology for estimating the most likely topology of a rooted, directed, three-dimensional tree given a single two-dimensional image of it. We regularize this inverse problem via a prior parametric tree-growth model that realistically captures the morphology of a wide variety of trees. We show that the problem of estimating the optimal tree has linear complexity if ancestry is known, but is NP-hard if it is lost. For the latter case, we present both a greedy approximation algorithm and a heuristic search algorithm that effectively explore the space of possible trees. Experimental results on retinal vessel, plant root, and synthetic tree datasets show that our methodology is both accurate and efficient.

Contents

List of Tables

List of Figures

Acknowledgements

First of all, I would like to thank my beautiful and supportive wife Helena for all her patience and help with this massive endeavor. I would also like to thank my parents for encouraging me to pursue my doctorate and for constantly being a source of support throughout my PhD. My deepest gratitude goes out to my PhD advisor Prof. Carlo Tomasi and to my collaborator and mentor Prof. Sina Farsiu. Their guidance has helped shape me as a researcher. I would also like to thank Prof. Scott C. Schmidler for his excellent advice on statistical topics and Profs. Ron Parr and Vince Conitzer for kindly reviewing my dissertation and providing feedback throughout these last few years.

1

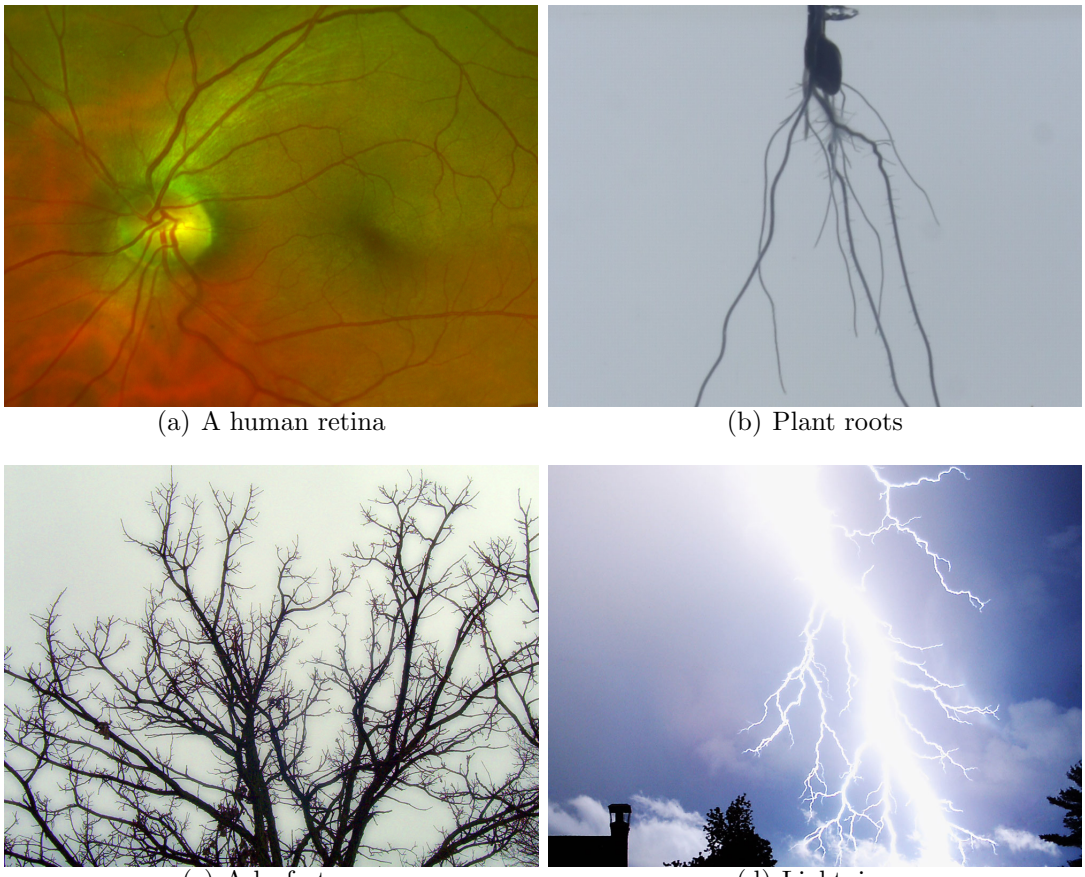
Introduction

Trees are fundamental physical structures in nature. Aside from the eponymous large plants, examples include retinal vessels, lung airways, neurons, lightning, plant roots, and more. Physical trees are typically the result of a branching process that grows away from an initial root to efficiently distribute a fluid, a current, or signals between a central source and a set of end-points. Different growth processes produce strikingly different trees, both in terms of their geometry and their connectivity, as Figure 1.1 illustrates.

A wide variety of imaging techniques—including fluorescein angiography, retinal fundus imaging, and x-ray and color photography—yield two-dimensional images of trees, from which it is often useful to reconstruct a tree's original connectivity. However, the three-dimensional location of each branch in the physical tree is lost in the projection, and parts of different branches often map to the same point in the image. See Figure 1.1 again.

Specifically, the image of a tree obscures its original topology in two key ways:

1. There may be spurious branch crossings in the image that resemble true branch-points.



(a) A human retina (b) Plant roots

(c) A leafy tree (d) Lightning

FIGURE 1.1: **Images of physical trees:** Different combinations of internal and external factors yield remarkably different trees. However, all these trees facilitate a hierarchical flow between a central node and a series of end-points. **(a)** and **(b)** are samples from our experimental datasets, while **(c)** and **(d)** are public domain images.

2. Information about the directionality (flow to or from the root) of the branches may be lost.

The resulting loss of information makes reconstructing tree connectivity and flow direction from an image an ill-posed problem, and prior models must be introduced to regularize the solution. Fortunately, good theoretical and empirical models have been developed in several domains to describe the expected morphology or growth pattern of a particular type of tree. Well-studied trees include blood vessels (Das et al., 2010; Macklin et al., 2009; Peirce, 2010; Perfahl et al., 2011; Quinas-Guerra

et al., 2012), plant roots (Band et al., 2012; Coelho et al., 2003; Dupuy et al., 2010), neurons (Koene et al., 2009; Larkman, 1991; López-Cruz et al., 2011), leafy trees (Bejan and Lorente, 2010; Reffye et al., 2012; Vos et al., 2010), and lightning (Gou et al., 2009; Lynn et al., 2012; Mansell et al., 2002).

In this work, we present a comprehensive methodology for estimating the most likely topology of a rooted, directed, three-dimensional tree given a single two-dimensional image of it and a growth model for that type of tree. We address this challenging inverse problem through a combination of greedy approximation and heuristic search algorithms that efficiently explore the space of possible trees. Our main theoretical contributions are as follows:

1. The formalization of the tree estimation problem from a single projection.

2. A parametric family of tree-growth models that realistically captures the morphology of a wide variety of trees.

3. A proof that the tree estimation problem is NP-hard if flow direction information is unknown.

4. A greedy linear-time algorithm for approximating the most likely topology of a tree.

5. A heuristic search algorithm that efficiently explores the space of possible trees starting from the greedy solution.

The rest of this work is organized into three parts. In Chapters 2 to 12, we present our tree estimation methodology and validate its effectiveness using various tree datasets. Then, in Chapter 13 we propose a number of potential improvements and extensions to our existing methodology. Finally, in Appendices A to C we give an overview of our graph extraction pipeline with which we estimate the initial graph from a noisy image.

In more detail, we first give an overview of related work on 3D tree reconstruction, 2D tree segmentation, and tree growth models in Chapter 2. In Chapter 3, we describe the geometry and combinatorics of projecting three-dimensional trees onto two-dimensional planar graphs, while in Chapter 4, we show how to partition branch crossings to convert the planar graph into a tree. In Chapter 5, we define how to consistently assign orientations to the graph's edges and in Chapter 6, we explore how many possible trees can project down to the same graph. In Chapter 7, we present a generative, parametric model of tree growth and in Chapter 8 we define the probability of an observed tree. In Chapter 9, we prove that an optimal tree estimate can be obtained in linear time when the flow direction in each projected branch is known, but is NP-hard to obtain when the direction is unknown. Given this hardness result, in Chapter 10 we introduce both a greedy approximation algorithm and a heuristic search method that refines the greedy solution. In Chapter 11 we develop several ways to quantify the similarity between two different trees. Finally, in Chapter 12 we present experimental results on retinal vessel, plant root, and synthetic tree datasets that show the effectiveness of our proposed algorithms.

The following chapter then explores further potential refinements and extension to our tree methodology. We first propose two ways of making our tree estimation more robust to errors in the input graph. Then, we present a novel sampling technique that can allow us to estimate a highly likely tree even if the initial greedy estimate is poor. Finally, we explore branch ordering and artery-vein classification, two extensions to our methodology which are relevant for retinal vessel applications. In Chapter 14 we conclude and review the presented work.

For completeness, in the appendices we present our pipeline for estimating the initial planar graph given an image. In Appendix A we show how to enhance the tree's branches and remove extraneous image artifacts. In Appendix B, we show how to segment the pixels corresponding to the tree from the pre-processed image.

Finally, in Appendix C we present a branch tracking algorithm that constructs the planar graph from the segmented image.

2

Related Work

The problem of estimating the most likely tree given a single two-dimensional image is novel, in the sense that there is almost no prior work that tackles this particular problem. The bulk of earlier tree estimation approaches have either made use of 3D data or multiple images, both of which allow the problem to be well-posed. In addition, there has been significant work on a number of related topics concerning trees, including growth modeling, image segmentation, and graphical modeling.

In this chapter, we will first review prior work on our tree estimation problem and then survey related work on tree imaging and modeling.

2.1 Prior work

To the best of our knowledge, the only prior work on automatically determining the topology of a three-dimensional tree given a single, two-dimensional image of it is the method developed by Zeng et al. (2006) for modeling visually plausible unfoliaged trees from images. Their algorithm greedily assigns a parent to each branch segment based on its relative angle with each candidate parent segment and the candidate's thickness. Unfortunately, the results presented in their paper were only qualitative

and did not include any quantitative evaluation.

2.2 Tree reconstruction

In tree reconstruction, the goal is to either segment out or estimate the topology of a tree from 3D volumetric data. Reconstruction from 3D data is a well-posed problem, since there is no dimension los. Therefore, most previous tree estimation work has focused on 3D reconstruction, primarily from CT and MRI scans of lung airways and cardiac vasculature.

2.2.1 Lung airways

The lung airways are part of the respiratory system. They distribute incoming air into millions of air sacs (alveoli) that then deliver oxygen into the bloodstream (Beers et al., 2004). The most popular methods in this field are region growing (Graham et al., 2008; Lo et al., 2010), and dynamic programming (Gülsün and Tek, 2008; Lo et al., 2009).

2.2.2 Heart and liver vasculature

The cardiovascular system consists of two vast trees rooted at the heart. Arteries carry oxygenated blood from the heart to the rest of the body, while the veins carry the deoxygenated blood back into the heart (Beers et al., 2004). Cardiac vasculature estimation has focused on probabilistic branch tracking (Friman et al., 2010; Schaap et al., 2007) and dynamic programming (Gülsün and Tek, 2008).

2.2.3 Plant roots

Roots are the part of a vascular plant that absorb water and other nutrients from the soil or other surrounding medium (Raven and Edwards, 2001). Tomographic reconstruction methods from conventional photographs have also been developed for

plant roots grown in a clear medium using volumetric carving (Iyer-Pascuzzi et al., 2013; Zheng et al., 2011).

2.3 Tree segmentation

Image segmentation is concerned with determining which pixels in an image correspond to the target object or objects. Tree segmetation seeks to extract the tree pixels but not the corresponding topology in space.

2.3.1 Retinal vessels

Most tree segmentation work has focused on retinal vessels. Retinal vessels supply blood to the retina. The retinal arteries and veins protrude from the optic nerve and innervate centrifugally to the retinal periphery (Hildebrand and Fielder, 2011). Segmentation methods employ local filtering (Xiao et al., 2013; Ricci and Perfetti, 2007; Soares et al., 2006), dynamic programming (Benmansour and Cohen, 2011; Estrada et al., 2012; Li and Yezzi, 2007), spanning tree sampling (Türetken et al., 2010, 2011), or tubular tracking (Pechaud et al., 2009; Yedidya and Hartley, 2008).

2.3.2 A/V estimation

Retinal vessels, being part of the cardiovascular system, are also subdivided into arteries and veins. There is a growing body of work on trying to distinguish these two types of vessels in the retina. Existing approaches have mainly used a combination of color features and vessel tracking (Perez et al., 2002; Rothaus et al., 2007; Dashtbozorg et al., 2013).

2.4 Graphical tree modeling

A related area in computer graphics is graphical tree modeling. Here, the goal is to efficiently generate realistic-looking trees. Work on modeling trees follows three

main approaches: In rule-based methods, trees are generated using fractals or other local deformations (Aono and Kunii, May; Prusinkiewicz and Lindenmayer, 1991). In sketch-based tree modeling, a user draws one or more 2D sketches and a tree is generated based on them (Chen et al., 2008; Ijiri et al., 2006; Tan et al., 2008). In image-based modeling, a set of input images is used to synthesize a plausible 3D tree model (López et al., 2010; Reche-Martinez et al., 2004).

The spirit of the above work differs significantly from ours. Graphical tree modeling focuses on generating visually plausible results. Our aim, on the other hand, is to estimate the actual topology of a given input tree faithfully.

2.5 Tree growth modeling

Finally, there has been considerable work in accurately modeling the growth of specific types of trees by accounting in detail for the interactions between the various forces that affect the growing tree. Here we review some representative works in these fields.

2.5.1 Blood vessels

Blood vessel models describe growth at multiple scales—from single cells to tissues—primarily through systems of differential equations that describe cell development and migration (Das et al., 2010; Macklin et al., 2009; Peirce, 2010; Perfahl et al., 2011; Quinas-Guerra et al., 2012).

2.5.2 Plant roots

Plant-root growth models generally employ similar principles, but also incorporate architectural constraints driven by gravity (Band et al., 2012; Coelho et al., 2003; Dupuy et al., 2010; Galkovskyi et al., 2012; Iyer-Pascuzzi et al., 2013).

2.5.3 Neurons

Neurons are electrically excitable cells that process and transmit sensory and motor information throughout the nervous system (Kandel et al., 2000). Neuron growth modeling has focused on statistical techniques, such as Bayesian networks (Koene et al., 2009; Larkman, 1991; López-Cruz et al., 2011).

2.5.4 Leafy trees

Leafy trees are large vascular plants with an elongated stem (Gschwantner et al., 2009). Leafy tree models have used flow diffusion and fractals (Bejan and Lorente, 2010; Reffye et al., 2012; Vos et al., 2010).

2.5.5 Lightning

Lightning are massive discharges of electricity between clouds or between a cloud and the Earth's surface (Rakov and Uman, 2003). Lightning modeling focuses on the ambient electric field (Gou et al., 2009; Lynn et al., 2012; Mansell et al., 2002).

In summary, there is a vast body of work on various topics related to tree analysis and modeling. In the following chapters, we will develop our own tree estimation methodology for the particular problem of determining the most likely three-dimensional tree given a single two-dimensional image.

3

Geometry and Combinatorics of Tree Projections

In this work, we study trees in three-dimensional space that project onto graphs in two-dimensional images, such as retinal vessels or plant roots. In general, our trees carry a flow of something—a fluid, a current, information, etc.—from their roots to their leaves or vice versa. Here, we assume that every branch transmits its flow in only one of these two possible directions. For simplicity, we assume that all flow is from the root, the reverse case being entirely equivalent.

We model physical trees as graphs embedded in three dimensions. More concretely, an *arborescence* is a directed, rooted tree in which there is a unique, directed path from the root to every other vertex in the tree. As Figure 3.1 shows, the arborescence's vertices and edges coincide with the physical location of the original tree's branches. Because they live in three dimensions and have a specific geometry, our trees are arborescences embedded in space.

We will have to distinguish between embedded and non-embedded directed trees. For clarity, we use the term "arborescence" when the embedding is included, and the term "*directed tree*" when it is not, even though "arborescence" is typically used for both in the literature. Both our arborescences and directed trees are rooted.

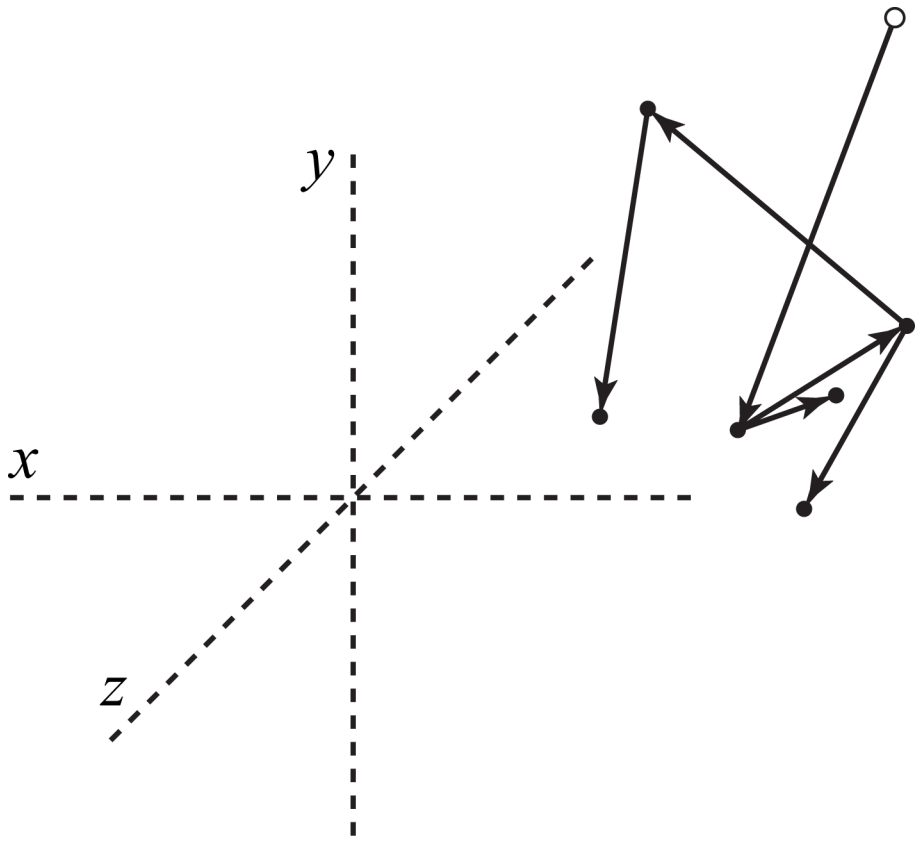


FIGURE 3.1: **Arborescences:** We represent a three-dimensional tree as a directed graph embedded in $\mathbb{R}^3$. The arborescence's vertices and edges are assumed to be connected and non-intersecting. The root is marked by a white dot.

Projection onto the 2D image plane obliterates information about the distance of an arborescence point from the plane, where the distance is measured along the point's projection ray. Most of the time, projection also obscures information about the direction of flow associated with each branch, as branches taper very slowly, if at all, in typical images. As a result, the projection of an arborescence is an *undirected graph* in the plane, once the arborescence has been segmented out from the image. This graph is planar if new branch intersections, formed as a result of projection, are defined to be new vertices. For brevity, we simply refer to "arborescences that project to graphs" in what follows, leaving qualifiers about direction, embedding, and dimension implicit.

Our task is to reconstruct a directed tree T (embedded in some arborescence A) from the graph G it projects onto. However, projection is a many-to-one mapping because of the loss of information it entails. As a result, many arborescences can project to the same graph, making the reconstruction problem ill-posed. To regularize it, we introduce a generative *prior model* M for the growth of any given type of tree—the dendrites of a neuron, the vessels in a retina, plant roots, or the branchings in a stroke of lightning. In our model, we induce a *prior probability* $p_M(A)$ on the set of all possible arborescences, and we then seek a most likely arborescence given the image graph and the model:

$$A^\star = \operatorname*{argmax}_{A \in \mathcal{A}(G)} p_M(A) \tag{3.1}$$

where $\mathcal{A}(G) \subset \mathcal{A}$ is the set of arborescences consistent with the graph G and $\mathcal{A}$ is the set of all arborescences.

The formulation (3.1) can be interpreted as a special case of Maximum A Posteriori (MAP) estimation,

$$A^\star = \operatorname*{argmax}_{A \in \mathcal{A}} p_O(G|A) p_M(A)$$

where the observation probability $p_O(G|A)$ is uniform (proportional to a constant) for graphs obtained by projecting A onto the image and zero otherwise. Thus, our formulation leaves all regularization up to the prior model M, while any potential noise in the image is handled in the preprocessing stage which extracts a clean graph G from the image.

In this work, we obtain a clean graph semi-automatically since current state-of-the-art graph extraction techniques, such as (Türetken et al., 2010, 2011; Yedidya and Hartley, 2008; Rattathanapad et al., 2012), are unfortunately not yet robust enough to consistenly estimate a correct planar graph from noisy images or images

that contain artifacts or very faint branches. Thus, our semi-automatic approach is as follows. We first extract a noisy graph as described in appendices A-C. Our automatic pipeline first enhances and then segments the tree's branches. It then extracts the planar graph from the segmented image. Finally, as described in Chapter 12, we manually correct any errors in the automatically estimated graph using a graph editing program we developed.

To address this limitation, in Chapter 13, we propose two ways of fully automating our tree estimation process. First, we outline a generalized observation model $p_O(G|A)$ which allows for errors in the planar graph and sketch out some strategies for optimizing this probability. Then, we also propose an extension of the approach presented in Appendix C that incorporates stronger shape and topology priors to more accurately estimate the initial planar graph.

The next three sections elaborate on the definition of the set $\mathcal{A}$ of all arborescences, on the notion of projecting an arborescence in space to a graph in a planar image, and on a technical refinement that simplifies further analysis. In the following chapters, we detail what happens at crossings between edges in the projection of an arborescence, describe criteria to determine when the orientations of a directed image graph are consistent with some arborescence in space, and count how many directed trees can project onto a given graph.

3.1 Arborescences

Let $A = (V_A, E_A, r_A)$ be a simple, finite, connected arborescence embedded in $\mathbb{R}^3$ and rooted at $r_A \in V_A$. By the definition of an arborescence, A is a tree, and the edges in E_A are oriented away from the root, that is, every edge

$$e = (u, v) \in E_A \text{ with } u, v \in V_A$$

is an ordered pair, with u the parent and v the child.

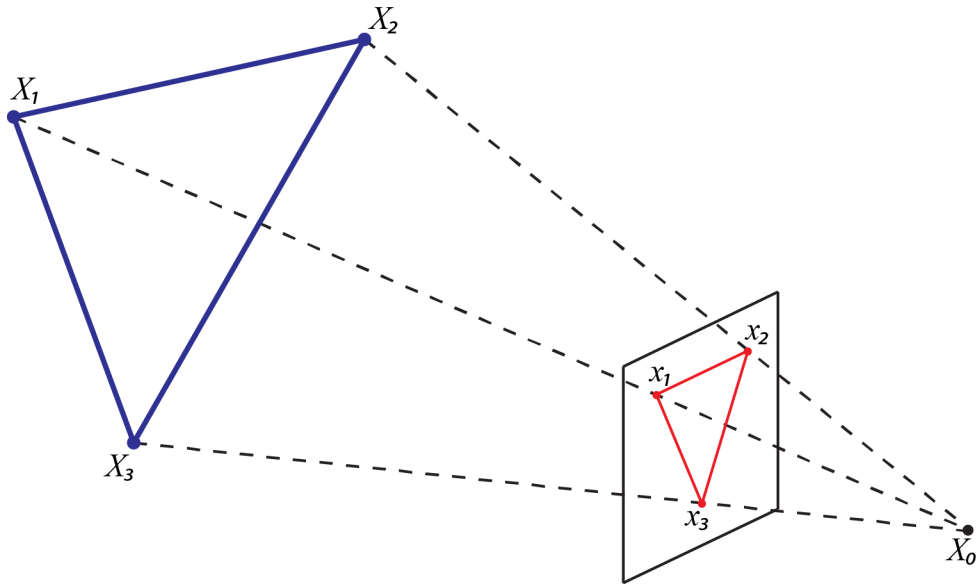


FIGURE 3.2: **Projection geometry:** Each world point X_i is projected to a corresponding point x_i on the image plane–indicated by the black square. The point X_0 is the center of projection and the projection lines are indicated by dashed lines. If the center of projection is at infinity, the projection lines are mutually parallel.

The arborescence A is *embedded in space*, in the sense that every vertex $v \in V_A$ maps to a point $\mathbf{v} = \eta(v)$ in $\mathbb{R}^3$ and every edge $e = (u, v) \in E_A$ maps to a simple, open, continuous curve $\mathbf{e}$ between the two points $\mathbf{u} = \eta(u)$ and $\mathbf{v} = \eta(v)$. In symbols,

$$\mathbf{e} \ : \ I \to \mathbb{R}^3 \ \text{ where } \ I = \{t \mid 0 < t < 1\}$$

is the open unit interval on the real line, and the following conditions hold:

$$\begin{aligned} &\lim_{t\to 0} \mathbf{e}(t) = \mathbf{u} \ \ , \ \ \lim_{t\to 1} \mathbf{e}(t) = \mathbf{v}, \\ &\text{and } t = t' \ \text{ if and only if } \ \mathbf{e}(t) = \mathbf{e}(t') \ . \end{aligned} \tag{3.2}$$

With this definition, the embeddings of vertices and edges are disjoint. We impose the further constraint that all vertices are distinct and all edges are mutually disjoint (Willard, 2004). In summary, if $\mathbf{u} = \eta(u)$ and $\mathbf{v} = \eta(v)$ are distinct vertices and $\mathbf{e}$ and $\mathbf{f}$ are distinct edges then

$$\{\mathbf{u}\} \cap \{\mathbf{v}\} = \mathbf{e}(I) \cap \mathbf{f}(I) = \{\mathbf{v}\} \cap \mathbf{e}(I) = \varnothing \ . \tag{3.3}$$

Since edge embeddings are simple and continuous and arborescences are connected trees, the limit conditions in (3.2) ensure that the embedding of the entire arborescence is continuous as well, in the sense that any two of its points are connected by an undirected path that is entirely on the embedding.

3.2 Projection

Let P be a projection from $\mathbb{R}^3$ to $\mathbb{R}^2$ (for example, an orthographic or perspective imaging projection). Geometrically, a set of points in $\mathbb{R}^3$ is projected onto an image plane by drawing *projection lines*, each through one of the points and a *center of projection*—a special, fixed point. The projection of this set is the set of points where the projection lines intersect the image plane. See Figure 3.2.

Projecting an arborescence onto a plane means projecting its embedding. This creates a planar embedding, which in turn induces a planar graph G that describes the topology of the planar embedding. For brevity, we refer to this process as simply "projecting an arborescence onto a graph." The resulting graph G is taken to be *undirected*, to capture the fact that projection typically obscures direction information.

More specifically, it will be useful for our discussion to separate the loss of one dimension in projection from the loss of direction information. To this end, we write the function from arborescence A to graph G in two steps:

$$G = U(G_d) \text{ where } G_d = P(A) \, . \tag{3.4}$$

In this notation, $P(\cdot)$ is the projection operator, G_d is the directed graph that A projects onto, and the operator $U(\cdot)$ deletes direction information. Since the graphs G and G_d differ only by the directions of their edges, these two graphs share the same set of vertices

$$V_G = V_{G_d} \, . \tag{3.5}$$

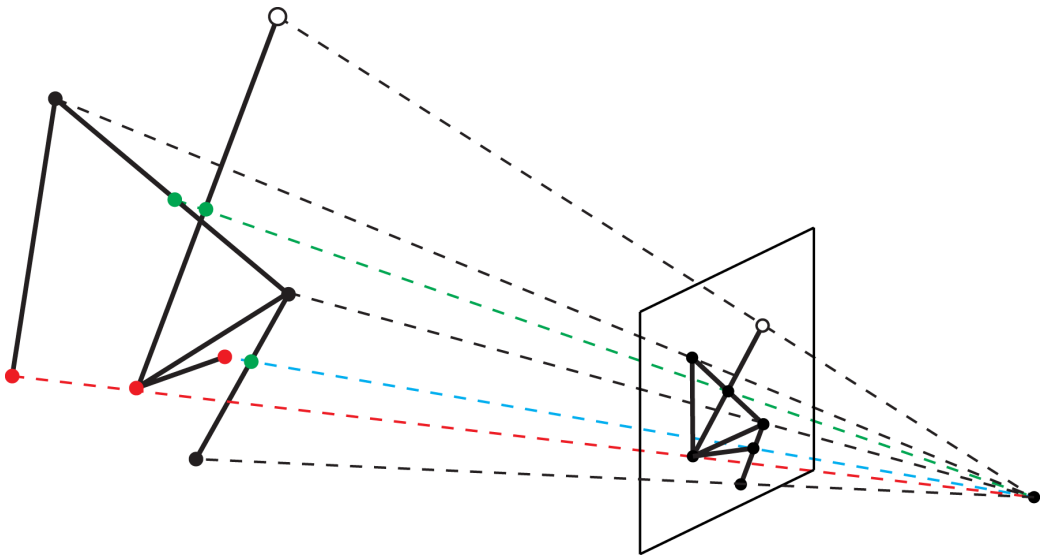


FIGURE 3.3: **Arborescence projection:** (This Figure is best viewed in color.) The projection of an arborescence onto a planar graph with circuits. The white dots mark the roots of both the arborescence and the projected graph. The green dots are the arborescence's refined vertices, which indicate the locations on the edges which cross with another point on the graph. Likewise, the red dots mark the vertices that cross with another point. The red, green and light blue projection lines indicate vertex-vertex, edge-edge, and vertex-edge crossings, respectively.

Both $P(\cdot)$ and $U(\cdot)$ are non-injective maps. Geometrically, if points in the embedding of arborescence A are slid along their projection lines towards or away from the center of projection while maintaining a continuous embedding for A, the projections G_d and G do not change. Because of this lack of injectivity from A to G_d to G, it is typically not possible to recover the full geometry of an arborescence from one of its projections, even if the projection were to preserve information about the direction of all edges.

However, both the directed graph G_d and its undirected version G preserve part of the geometry and topology of A. More specifically, any arborescence A that projects to G_d and to G must be consistent with their embeddings. In addition, the directions of the edges in A must be the same as those in G_d and the edge-vertex incidence relations in A must be compatible with those of G_d and G, as defined in the next

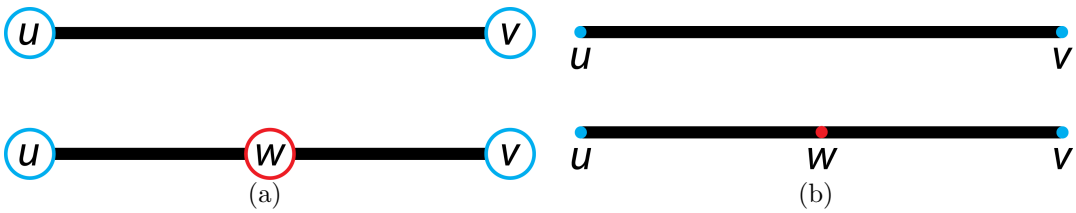


FIGURE 3.4: **Edge subdivision:** In **(a)**, an edge subdivision replaces a single edge $[u, v]$ with two new edges $[u, w]$ and $[w, v]$. In **(b)**, the same subdivision in an embedded edge maintains the continuity of the embedding. Old vertices are highlighted in light blue, while the new vertex is shown in red.

subsection.

3.3 Refinement

When an arborescence projects onto a plane, multiple vertices can project to the same point, distinct edges can intersect in the projection, and vertices can project onto edges, as Figure 3.3 illustrates.[1]

In order to simplify our analysis, we now *refine* an arborescence A that projects to a directed graph G_d into a new arborescence A' by adding new vertices to edges of A so that only vertices collide when A' projects to G_d.

We do this by *subdividing* every edge of A at every crossing in G_d. The subdivision of an edge $e = (u, v)$ in an arborescence—or indeed any directed graph—splits edge e into two by adding a new vertex w to the vertex set V_A, removing the old edge e from E_A, and adding new edges (u, w) and (w, v) to E_A.

In an embedded graph, the subdivision of edge e induces a continuous subdivision of its embedding $\mathbf{e}$ in the obvious way: by re-defining a point along $\mathbf{e}$ as a new vertex $\mathbf{w} = \eta(w)$. See Figure 3.4. Because of continuity, the embeddings of the two arborescences A and A' have identical images, and the two graphs are homeomorphic (in the graph-theoretical sense).

[1] Additionally, edges can overlap over line segments or project to a single point, but we will not consider these cases in our analysis. See (4.1) in Section 4.1.

If every edge of A that intersects another edge or crosses a vertex when projected onto G_d is subdivided where the collision occurs, the resulting arborescence A' is said to be G_d-*refined*. In a G_d-refined arborescence, the only possible collisions are between vertices. Since G_d and its undirected version G share the same vertex set (see (3.5)), G-refinement and G_d-refinement are equivalent operations.

Due to possible collisions, the number of vertices in G_d is at most equal to the number of vertices in A'. Thus, given also (3.4) and (3.5), we have

$$|V_G| = |V_{G_d}| \leqslant |V_{A'}||E_G| = |E_{G_d}| = |E_{A'}|.$$

Here, $|V|$ denotes the size of set V. By definition, since the number of edges remains constant, if $|V_G| < |V_{A'}|$ then G will contain at least one circuit and will thus not be a tree.

In the next chapter, we will further explore how these intersections affect the properties of the projected graph.

4

Crossings and Valid Partitions

In the previous chapter, we defined how to project an arborescence A embedded in three-dimensions onto a plane. In general, the resulting planar graph G is not necessarily a tree, however, since multiple points on the arborescence can project to the same point on the plane. Here, we will elaborate on how these many-to-one projections affect the properties of the resulting planar graph and how to undo the effects of intersections between projected vertices.

4.1 Crossings

Let X be the set of points in a directed graph G_d or its undirected version G that are formed by the projection of multiple vertices in some refined arborescence A. We call these points *crossings*:

$$X = \{\mathbf{x} \in \eta(G) \text{ such that } |P^{-1}(\mathbf{x})| > 1\} \ .$$

With slight abuse of notation, we do not distinguish between graph vertices and their embeddings when using the term "crossing." As discussed in Section 3.3, crossings are the only possible collisions when a refined arborescence projects onto a plane. In this

work, we assume that the root is not a crossing. Furthermore, we assume that every pre-image $P^{-1}(\mathbf{x})$ for any point $\mathbf{x}$ on the embedding of graph G is finite—whether $\mathbf{x}$ is a crossing or not—and that the number of crossings is finite:

$$\begin{aligned} |P^{-1}(\mathbf{x})| &< \infty, \quad \forall \mathbf{x} \in \eta(G) \\ |X| &< \infty. \end{aligned} \tag{4.1}$$

The first condition excludes any segment of an edge from projecting to a single image point, and the second implies that any two projected edges intersect at a finite number of image points.

As shown in the previous chapter, if a projected graph has at least one crossing then it will not be a tree because it will have fewer vertices, but the same number of edges as the refined arborescence. Furthermore, we now show that under the above assumptions:

Proposition 1. *Crossings are exactly those vertices with in-degree greater than 1 in the directed graph G_d.*

Proof. Let v be a vertex of a planar, directed graph G_d, and let $\deg(v)$ denote the degree of v. Since G_d is the projection of some refined arborescence A, each edge entering v must enter a distinct vertex in A—or else A would not be a tree. Thus, if $\deg^-(v)$ of the $\deg(v)$ edges incident to v are directed into v, there must be exactly $\deg^-(v)$ vertices in A that project to v. In other words, the multiplicity of v is equal to its in-degree $\deg^-(v)$.

Since crossings correspond to multiple vertices in a refined arborescence, they are exactly the vertices of G_d for which $\deg^-(v) > 1$. No vertex other than the root can have in-degree zero. □

In summary, projection fuses two or more vertices at each crossing, thereby obscuring the original topology of the tree. In order to estimate the tree's topology

given the projected graph, we need to be able to reverse this fusion by *partitioning* the edges incident to each crossing, as described next.

4.2 Valid partitions

In a crossing v, each edge entering v corresponds to a different vertex in A. Therefore, each of the $\deg^+(v) = \deg(v) - \deg^-(v)$ edges exiting v is the projection of an edge exiting one of these vertices. Since A is a tree, each exiting edge of v, if any, can have only one parent edge; thus they are partitioned into $\deg^-(v)$ sets. A partition of the edges incident to v, in which each edge exiting v is associated with exactly one edge entering v, is called a *valid partition* of v.

In more detail, let $E_-(v), E_+(v) \subseteq E(G_d)$ be the sets of edges entering and exiting v, respectively. Here, we assume both sets are ordered arbitrarily. Then, we define $S(v) = \{S_1, S_2, \cdots S_{\deg^-(v)}\}$ to be an assignment of exiting edges to entering edges such that:

$$S_1 \cup S_2 \cup \cdots S_{\deg^-(v)} = E_+(v)$$

$$S_i \cap S_j = \varnothing, \quad \forall S_i, S_j \in S(v).$$

Thus, $S(v)$ is collectively exhaustive over $E_+(v)$ and its subsets are mutually disjoint.

Now, let W be a set of $\deg^-(v) - 1$ new vertices, such that $V(G_d) \cap W = \varnothing$. Then, a valid partition on v with assignment $S(v)$ is a transformation from G_d to a new graph

$$G'_d = \text{partition}(G_d, v, S(v)) = (V', E')$$

such that

$$V' = V(G_d) \cup W$$

$$E'_+(v) = S_1 \quad \text{and} \quad E'_-(v) = e_1^-(v)$$

$$E'_+(w_i) = S_i, \quad \text{and} \quad E'_-(w_i) = e_i^-(v) \quad \forall w_i \in W,$$

where $e_i^-(v)$ is the i-th edge entering v and $E'_-(v)$ and $E'_+(v)$ are the new sets of entering and exiting edges of v in G'_d. Note that, since $S(v)$ is unordered, the choice of S_1 is arbitrary. Each subsequent $E'_+(w_i)$ corresponds to one of the other subsets of exiting edges of $S(v)$.

In words, a valid partition adds $\deg^-(v) - 1$ new vertices W to the graph and reassigns

$$\deg^-(v) - 1$$

of the entering edges of v to enter each of the new vertices. It then reassigns zero or more of the edges exiting v to each w. Everything else stays the same. Figure 4.1 illustrates all the possible valid partitions for a sample vertex. By construction,

$$|V'| = |V(G_d)| + \deg^-(v) - 1 \tag{4.2}$$

$$|E'| = |E(G_d)| \ . \tag{4.3}$$

This implies that partitions are *monotonic*, in the sense that no partition can undo another partition. Also, since $\deg^-(v) - 1 \in [2, |E_d(v)|]$, the number of vertices cannot decrease. Furthermore, valid partitions cannot create new paths between existing vertices in a graph, because they only separate edge endpoints, and do not merge any.

Partitions are *local*, in that they leave most of the graph unaffected. Given a vertex $v \in V(G_d)$, let $N(v) \subseteq V$ be the set of immediate neighbors of v:

$$N(v) = \{u \mid \{u, v\} \in E\},$$

and let

$$N_+(v) = N(v) \cup \{v\} \ .$$

Then, since a partition of vertex v only involves edges incident to v, all edges incident only to vertices in $V \backslash N^+(v)$ are unchanged.

As we show in Chapter 9, locality separates the effects of different partitions from each other, and leads to an efficient way to transform a graph into a tree by multiple partitions.

Two valid partitions $(u, S(u))$ and $(v, T(v))$ of the same graph are *distinct* if they either partition different vertices or correspond to different partitions of the vertex's exiting edges:

$$u \neq v \;\text{ or }\; S(v) \neq T(u)\ . \tag{4.4}$$

Furthermore:

Proposition 2. *Distinct partitions* $(u, S(u))$ *and* $(v, T(v))$ *are* unique*, in the sense that they yield different graphs:*

$$\text{partition}(G_d, u, S(u)) \neq \text{partition}(G_d, v, T(v))\ .$$

Proof. If $u = v$, the condition in (4.4) ensures that the partitions of the edges in the two cases are different, so the resulting graphs are not identical to each other.

If on the other hand $u \neq v$, then a partition of u separates $E(u)$ into two or more nonempty sets. No partition of v can do that, so the two resulting graphs must again be different. □

Finally, (4.2) shows that applying a valid partition strictly increases the number of vertices while keeping the number of edges constant. By definition, a tree is a connected graph such that $|V| = |E| + 1$. Therefore, if we apply a sequence of partitions to a graph, such that no partition disconnects the current graph, then the sequence is guaranteed to converge to a tree. In the next chapter we will explore further properties of the initial directed graph G_d that will allow us to determine when a sequence of splits will converge to a tree.

4.3 Crossing degree distribution

To conclude our discussion on crossings, we will analyze the expected degree of a crossing; that is, how many incident edges it is likely to have. So far, we have assumed that P is an *idealized* projection in the sense that two points in A may project arbitrarily close to one another and still be distinguished from one another in G and G_d. Actual imaging systems, on the other hand, such as fundus cameras or x-ray, have limited resolution. In the latter case, two points whose projections lie closer than the point spread function (PSF) of the imaging system will be indistinguishable in the resulting graph.

The distribution over the possible degrees of a crossing in these two scenarios can be very different. In an idealized projection, each crossing almost surely[1] has degree four since crossings of higher or lower degree require two or more vertices to project to the same point on the plane (as opposed to two edges). In a limited-resolution system, on the other hand, the probability of higher- and lower-degree crossings is non-zero, as shown in our experimental data in Chapter 12. In the latter case, a set of branch-points and end-points whose projections are close to one another will be merged into a single, higher-degree vertex in the resulting graph.

[1] in the measure theory sense (Rogers and Williams, 2000).

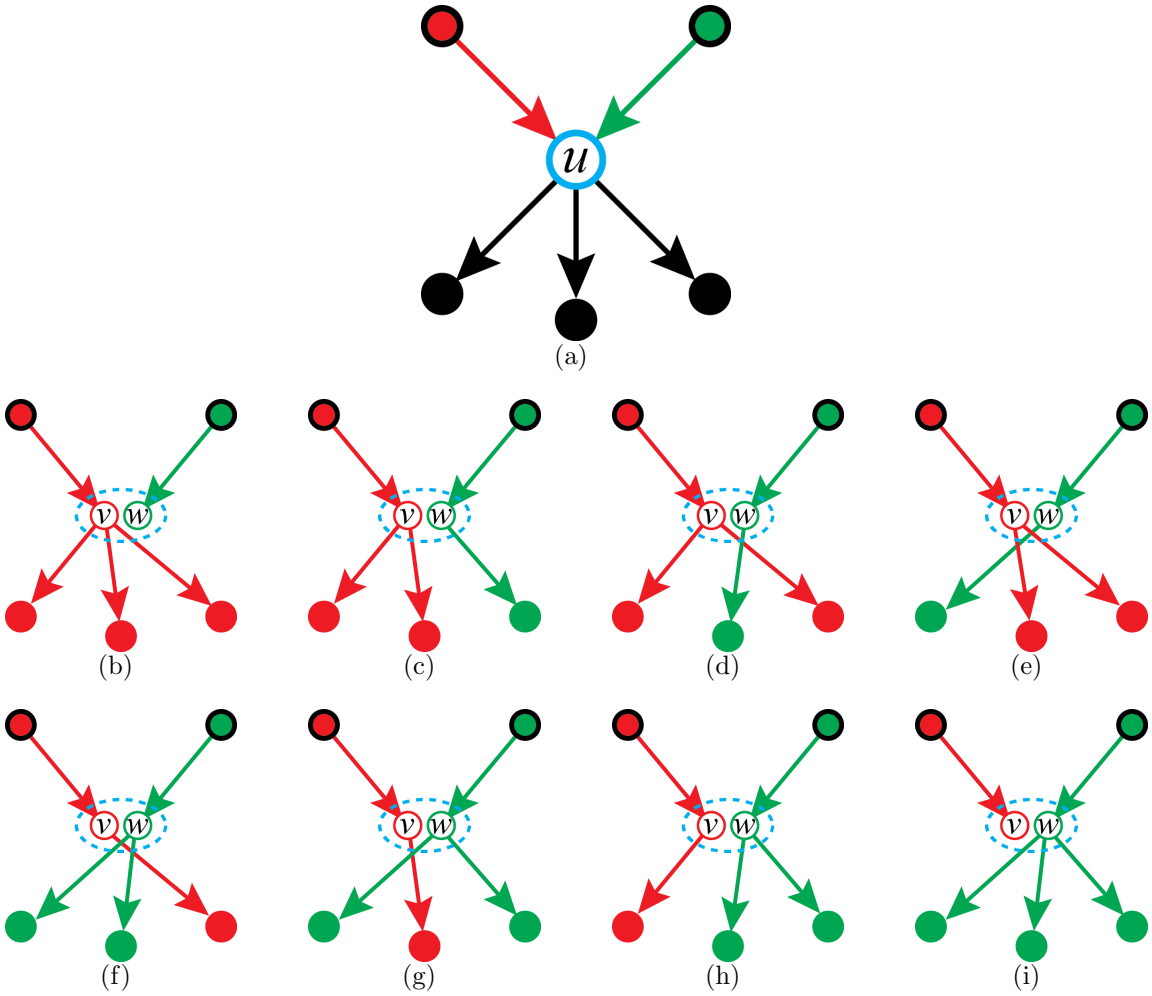


FIGURE 4.1: **Valid partitions:** (This Figure is best viewed in color.) **(a)** A vertex u and its neighborhood in a directed graph; u has two incoming edges, highlighted in red and green, and three outgoing edges. The eight possible valid partitions of u are shown in **(b-i)**. In each case, u is partitioned into two vertices v (in red) and w (in green) such that each vertex has a single parent and then each of the outgoing edges of u is assigned to one of the new vertices. In the middle row v has either two or three children, and in the bottom row it has zero or one (and vice-versa for w). The dotted oval indicates that v and w share the same location on the plane.

5

Flow Orientations

The previous chapters explored the effects of projection on the geometry and connectivity of arborescences embedded in three-dimensional space. This chapter focuses on edge orientation. As noted in Chapter 1, a projection often obscures the original direction of flow along the different edges of the graph.

Specifically, we define the two sets of all arborescences that are consistent with either a directed graph G_d or an undirected graph G, respectively, and relate these two sets. Given an undirected graph G, we also determine which of its *orientations*—ways to assign a direction to each of the edges in G—are consistent with some arborescence. In Section 5.2, we show that all connected, undirected graphs can be oriented so as to be consistent with some arborescence—and typically with many of them. In Section 5.3 we introduce a way to "walk" between all appropriate orientations of a graph G by the simple operation of flipping a single edge orientation at a time. This result will be crucial to our ability to search the space of all appropriate orientations in search of a good tree (Chapter 10).

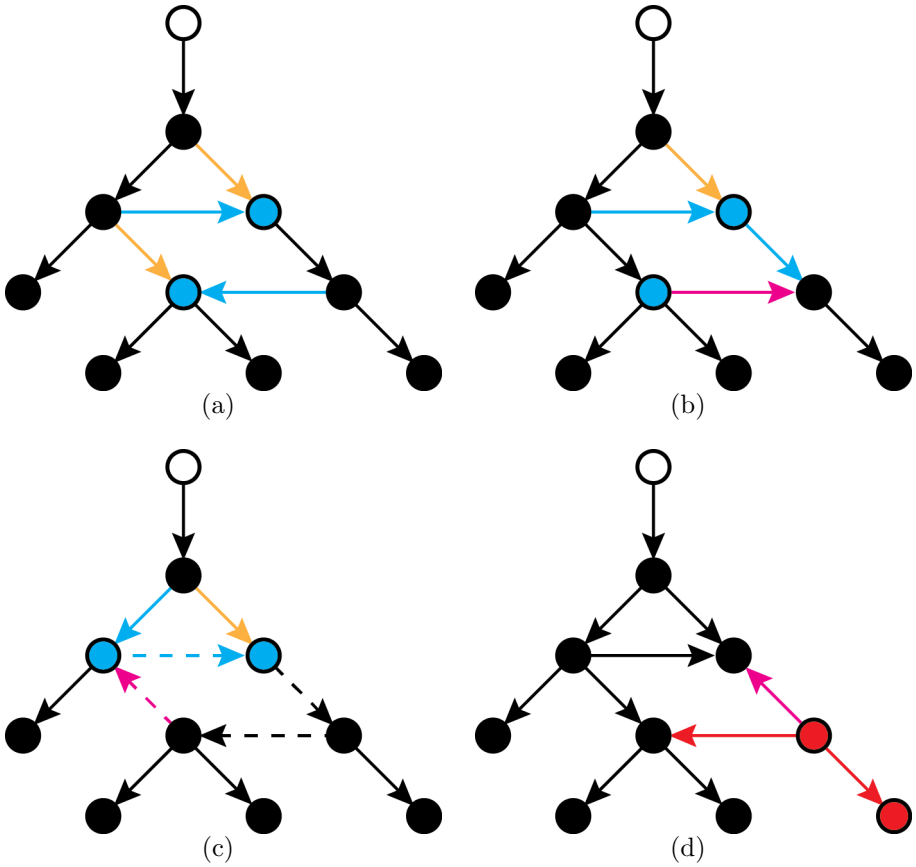


FIGURE 5.1: **Flow orientations:** (This Figure is best viewed in color.) **(a)** An acyclic flow orientation of a planar graph. The root is marked by the white dot. The crossings are marked in blue. Flipping the blue edges yields another flow-dag **(b)**, while flipping the yellow ones gives a cyclic flow orientation **(c)**, and flipping the black edges produces a non-flow orientation **(d)**. In (b-d), the flipped edge is highlighted in magenta. The dashed edges in (c) indicate its directed circuit, while the red edges and vertices in (d) indicate the subgraph that is unreachable from the root.

5.1 Flow orientations

Let G_d be a directed, planar graph. We define a *G_d-tree* to be a directed, rooted tree T which can be embedded in a refined arborescence A that projects onto G_d. The set of all G_d-trees is denoted by $\mathcal{T}(G_d)$. Similarly, given an undirected, planar graph G, the class of all directed, rooted trees for which there exists a refined arborescence A that projects onto G is denoted by $\mathcal{T}(G)$. The elements of $\mathcal{T}(G)$ are called *G-trees*.

Not all directed, planar graphs have G_d-trees. To see this, recall that every vertex

in an arborescence A is reachable through a directed path from the root r_A of A. Since the projection operator $P(\cdot)$ in (3.4) does not eliminate any paths—directed or otherwise—this reachability property must hold for G_d as well, although it is possible that crossings introduce additional paths from the projection r_{G_d} of r_A to some of the vertices of G_d. We say that a directed graph G_d is a *flow orientation* if this reachability property holds for some root; that is, if there exists a vertex r_{G_d} from which every vertex in G_d is reachable through at least one directed path in G_d. More concretely:

Proposition 3. *A directed graph G_d admits at least one G_d-tree if and only if G_d is a flow orientation.*

Proof. First if a given directed graph G_d is not a flow orientation, then G_d has no G_d-tree since at least one vertex is unreachable from the root. Conversely, if G_d is a flow orientation, then one can construct a G_d-tree by first using a spanning tree algorithm to build a directed spanning tree rooted at r_{G_d}. Then, each of the edges of G_d that are not in the directed spanning tree is incident to two vertices in it, and can be attached to its parent in G_d. The resulting graph is still a tree and includes all the edges of G_d. □

We will now explore further properties of flow orientations. If the root r_G of a given undirected graph G is kept fixed and G happens to be a tree, then G has a unique flow orientation, in which all edges are oriented away from r_G. Thus, in order to admit multiple flow orientations, the graph G rooted at r_G must have circuits. Furthermore, we will now show that:

Proposition 4. *If we assume that the root r_G is not a crossing, then r_G must have in-degree zero in all possible flow orientations rooted at it.*

We will show this by proving the equivalent statement: *the root of a flow orientation is a crossing if and only if it has positive in-degree.*

Proof. We will first prove that if the root r_G is a crossing, then it must have positive in-degree. By definition, the root r_T of any directed tree consistent with G_d projects down to r_G. If r_G is a crossing, then at least one other vertex $v \in V_T$ projected down to the same location. The last edge in the path from r_T to v is incoming to r_G, so r_G has positive in-degree if it a crossing.

To prove the reverse direction, note that all paths to every vertex in a flow orientation rooted at r_G start at r_G. If the root has positive in-degree, then at least one path must flow back into the root. Such a path forms a directed (and undirected) circuit C in the flow orientation. To transform the flow orientation into a tree, we have to remove this circuit. We will now show that if a flow orientation contains one or more directed circuits that include the root, then, to obtain a tree, we will be forced to split the root to break up one at least one of these directed circuits.

We will first address the case in which there is only one directed circuit that includes the root:

(i) *Let G_d be a flow orientation with a single, directed circuit C that includes r_G. Then, the graph resulting from splitting C can only be a tree if and only if we split at the root.*

To see this, assume that we split C at a vertex $v \in C$ that is not the root. Since G_d had only one directed circuit that included the root, then the path from r_G to v in C was the only directed path to v in G_d. In this case, after the split there is no directed path from the root to v, so the resulting graph cannot be a directed tree. On the other hand, if we split at the root, we can always obtain a tree by creating a leaf at the end of the edge incoming to r_G.

In other words, if G_d contains only one directed circuit, then no vertex in it, other than root, can be a crossing. Thus, in case (i) the root is forced to be a crossing.

Now, let G_d be a flow orientation with k directed circuits that include r_G. We will now show that no matter how we split the first $k-1$ circuits, we will always find

ourselves in case (i):

(ii) *Let v be a crossing, which is not the root, which is part of one or more directed circuits that include the root. Then, all possible valid partitions of v result in a graph that still has at least one directed circuit.*

To see this, let e be an outgoing edge of v which is part of directed circuit that includes the root. In all possible valid partitions of v we have to assign e to one of the incoming edges of v. However, all these incoming edges are part of a path from the root to v. Thus, no matter which incoming edge we pick, there will still be directed path from the root back to itself.

Therefore, even if we only partition crossings, since we have to maintain the property that all the vertices that result from the partition are still reachable from the root, we will always leave at least one directed circuit untouched unless we split the root, as in case (i). Therefore, the root must be a crossing if and only if it has positive in-degree. □

5.2 Acyclic flow orientations

Each G_d-tree in $\mathcal{T}(G_d)$ corresponds to a different combination of valid partitions for each of the crossings of G_d. However, not all these combinations yield a G_d-tree. Specifically:

Proposition 5. *If G_d contains directed circuits, then at least one choice of valid partitions will result in a graph that is not a G_d-tree.*

Proof. Let V_C be the set of crossings that are all part of one of the graph's directed circuits. By construction, $|V_C| \geqslant 1$: In a flow orientation, there is at least one path that connects every directed circuit to the root, and the node where that path meets the circuit is a crossing. For every $v_i \in V_C$, let u_i and w_i be the parent and child of v_i that are also part of the circuit. Then, partition each v_i such that u_i is assigned only

w_i. This construction detaches the circuit from any path into it, and the resulting graph is disconnected. □

Conversely, if G_d is both a flow orientation and a directed acyclic graph—or *flow-dag*—then:

Proposition 6. *Every possible combination of valid partitions for each of its crossings yields a G_d-tree.*

Proof. Let G_d be a flow-dag, let v be any of its crossings, and let V_I and V_O be the sets of parents and children of v, respectively. In a dag, no vertex can be its own ancestor. Thus, every directed path from the root to V_I cannot include v or any of the vertices in V_O, so no matter how we associate the vertices in V_O to those of V_I, there will still be a directed path from the root to both V_I and V_O. Therefore, regardless of how we partition v, we will not affect the reachability of any vertex. □

In the rest of this work, we will restrict our focus to flow-dags. Fortunately, it easy to show that:

Proposition 7. *All connected, undirected graphs admit at least one flow-dag orientation.*

Proof. Define a total order of its vertices, start at the root, and orient every edge from the earlier to the later end-point in the ordering. □

5.3 The flow-dag meta-graph

Given any orientation of a graph, we can obtain a new orientation by changing the direction of—or *flipping*—one of its edges. However, not all edge flips starting from a flow-dag G_d produce another flow-dag. Specifically:

Proposition 8. *Flipping edge e in G_d yields a flow-dag G'_d if and only if the following two conditions hold:*

- *The edge e enters a crossing v of G_d.*

- *The directed graph resulting from the flip is acyclic.*

Proof. The second condition can be verified by a depth-first search (Tarjan, 1976). To justify the first condition, we now argue that G'_d is a flow orientation if and only if e enters a crossing. If the second endpoint v of the directed edge e is a crossing, then the in-degree of v is greater than 1. Because of this, flipping e still leaves vertex v reachable from r through one of its other incoming edges and does not disconnect any other vertex from the root. Conversely, if v is not a crossing, flipping its sole incoming edge will disconnect it from the root. □

Thus, the two conditions above are both necessary and sufficient for G'_d to be a flow-dag, and G_d and G'_d are declared to be *neighbors* of each other if and only if both conditions hold. This neighborhood relationship induces a graph over all flow-dags. To avoid confusion with other graphs, we call this the flow-dag *meta-graph* (see Figure 5.2).

We now show that the flow-dag meta-graph is connected. That is:

Proposition 9. *We can transform any flow-dag for a graph G into any other by repeatedly flipping one edge such that every intermediate orientation is also a flow-dag.*

Proof. We will show the above property by relying on a similar property that links all the directed spanning trees of a graph.

Adding one edge to a spanning tree necessarily creates a cycle, called a *fundamental circuit*. Two spanning trees are said to be neighbors of each other if and only if they differ by swapping edges in a fundamental circuit. Specifically, a spanning tree is obtained from one of its neighbors by removing one edge from the latter and replacing it with another edge from the same fundamental circuit. This operation

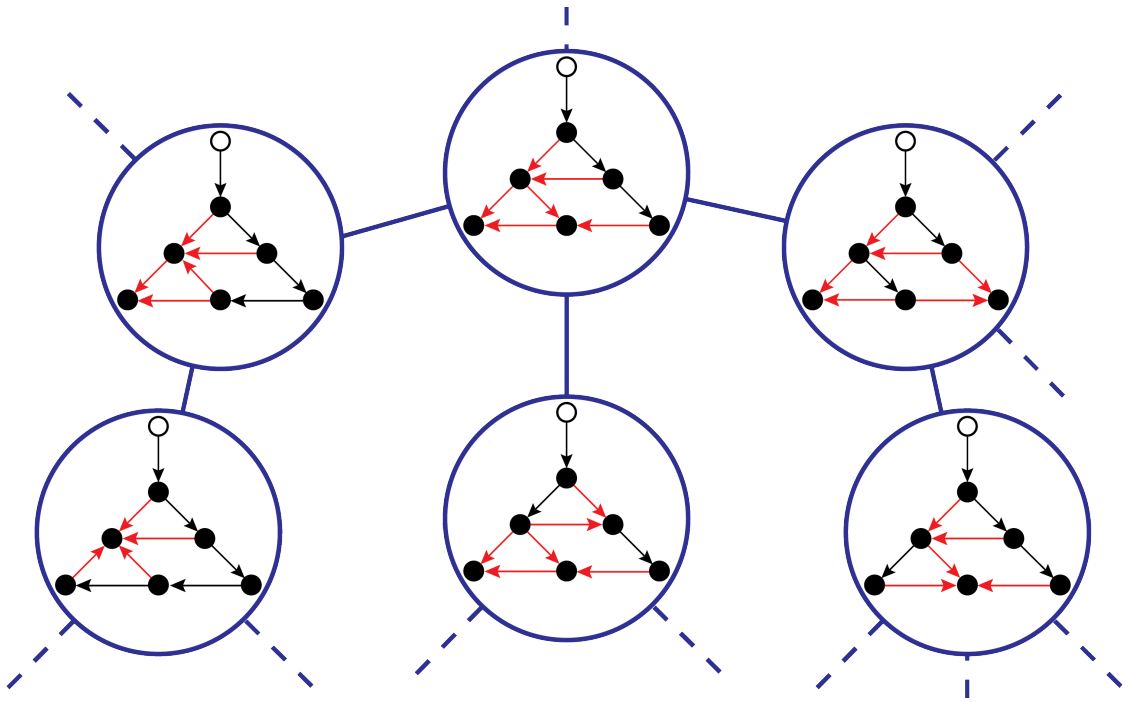


FIGURE 5.2: **The flow-dag meta-graph:** (This Figure is best viewed in color.) The flipping operation induces a connected graph over the set of acyclic flow orientations of a graph. The meta-graph's vertices and edges are shown in blue. In each flow-dag, the edges incoming to a crossing are shown in red. Neighboring orientations differ by only one flip.

maintains the spanning tree property if the new edge connects to the new spanning tree the same vertex v that the old edge used to connect to the old spanning tree. In other words, the directed edge (u, v) being replaced shares child v with the new edge (w, v). It has been shown (Kapoor and Ramesh, 2000) that repeated application of this replacement operation can transform any directed spanning tree into any other such that every intermediate graph is itself a directed spanning tree. As a consequence, the space of (directed) spanning trees of a given graph is connected through edge swaps.

To extend neighborhood from spanning trees to flow-dags, we say that a flow-dag G_d for an undirected graph G is *consistent* with a directed spanning tree D of G if the orientations of the edges in D coincide with those of the edges in G_d. We then show that (i) the flow-dags consistent with a given, directed spanning tree for G are

neighbors of each other, and that (ii) for each pair of neighboring spanning trees, there is a flow-dag that is consistent with both of them. These two facts, together with the neighborhood property for spanning trees, ensure that we can walk from flow-dag to flow-dag by flipping edges.

To prove property (i), let D be a directed spanning tree of G and let $E_{\setminus D}$ be the set of edges of G that are not in D. Let (u, v) be one possible orientation of an edge $\{u, v\} \in E_{\setminus D}$. The edge (u, v) is a *forward* edge (w.r.t. D) if u is an ancestor of v in D, a *backward* edge if v is an ancestor of u, or a *cross* edge otherwise. A backward edge always creates a directed circuit, so it cannot be part of any acyclic orientation of G. Thus, it is not possible to flip a forward edge, since it would create a backward edge. On the other hand, a cross edge can be oriented in either direction; each choice corresponds to a different total ordering of the two vertices. Now, let G_d and G'_d be any two flow-dags that contain D. Since both orientations are flow-dags, the reasoning above shows that all the edges on which they differ are cross edges. Thus, we can convert one orientation to the other by flipping each of these edges one at a time.

We now show property (ii): any two neighboring spanning trees share at least one flow-dag. Let D' be a neighbor of D; that is, D' differs from D by replacing edge $e = (u, v)$ with edge $e' = (w, v)$ that shares the same child v with e. Edge e' cannot be a backward edge (w.r.t. D) since if v is an ancestor of w in D, then there is no path from the root to v in D'. Then, let G_d be a flow-dag that contains both D and e'. By construction, G_d also includes D', so G_d is consistent with both D and D'.

In summary, we can walk between any two flow-dags G_d and G'_d associated with an undirected graph G with root r_G as follows: Construct a spanning tree D consistent with G_d and a spanning tree D' consistent with G'_d. Use the spanning tree neighborhood property to construct a sequence of spanning tree neighbors that connect D to D', and a corresponding sequence of flow-dags that connect consecutive

spanning trees in the sequence (by property (ii)). These flow-dags connect sets of flow-dags corresponding to spanning trees in the sequence. Since flow-dags in these sets are mutually connected by edge flips (by property (i)), the two flow-dags G_d and G'_d are connected to each other as well, and the flow-dag meta-graph is connected. $\square$

In the next chapter, we will explore bounds on the number of directed trees that can project to a given planar graph.

6

The Number of Tree Solutions

In this chapter, we prove that under mild assumptions the number of G-trees for a given graph G is finite. However, we also prove that this number can be very large—more precisely, it is exponential in the number of faces of the planar graph G. We will then show that finding *some* G-tree consistent with a graph G is trivial. On the other hand, finding a *realistically plausible* G-tree consistent with G and with some model of growth for the type of arborescences under consideration is an entirely different matter, which will be taken up in Chapter 8.

Here, we assume the constraints listed in (4.1). We also assume that all graphs G obtained by projecting arborescences onto planes have bounded degree. That is, there is a fixed integer $c > 0$, independent of the number of vertices in the arborescence, such that no vertex in G is incident to more than c edges. This assumption is trivially valid for any one graph one can detect in an image and is used only in considerations of asymptotic complexity, which we explore further in Chapter 9. Under the above constraints:

Proposition 10. *The number of G-trees for a given graph G is finite.*

Proof. Given a graph G rooted at r_G, one can construct a G-tree T by making the following choices:

1. Assign multiplicities to each vertex of G. That is, specify how many vertices of T project onto each vertex of G.

2. For each vertex v in G with multiplicity greater than 1, assign each of the edges that are incident to v to one of the vertices of T that project to v.

The multiplicity of a vertex of G cannot be greater than the number of edges incident to it, so each multiplicity is bounded, and there is a finite number of ways to allocate multiplicities (step 1). Similarly, there is a finite number of assignments of edges to vertices (step 2).

Not all choices lead to valid G-trees, because the resulting graph may not be a tree. Nevertheless, the number of choice combinations outlined above is an upper bound on the number of G-trees, so the number of G-trees for a given graph G is finite. □

Figure 6.1 shows the results of some of these choices for a small graph G. One could then root the resulting tree T by setting $r_T = r_G$, and orient the edges of T away from r_T to obtain an arborescence.

It is easy to find one G-tree given a graph G. To this end, pick a vertex r_G of G and map that to the root of the output G-tree. Then make a spanning tree for G starting at r_G, directing each edge consistently away from r_G. This covers all of G except for edges that would close fundamental circuits, called *non-tree edges*. Connect these edges to the spanning tree at either end—arbitrarily or according to a topological sorting—and orient them consistently away from r_G. The resulting graph is a G-tree.

The construction above also shows that *the number of G-trees is large.* A lower bound for their number can be found by noting that a planar graph (such as G)

can have a number of spanning trees that is exponential in the number of vertices (Buchin and Schulz, 2010). In addition, each spanning tree leaves out $|E| - |V| + 1$ non-tree edges (since it has $|V| - 1$ edges), each of which admits two orientations. So the worst-case number of G-trees is at least exponential in the number of vertices.

While we have just shown that finding *some* tree that projects to a given graph G is easy, determining *the most likely* tree in $\mathcal{T}(G)$ given a growth model for a certain class of trees is an entirely different matter. The next section introduces a simple but versatile arborescence growth model, and shows how to use the model to rate the likelihood of any given tree.

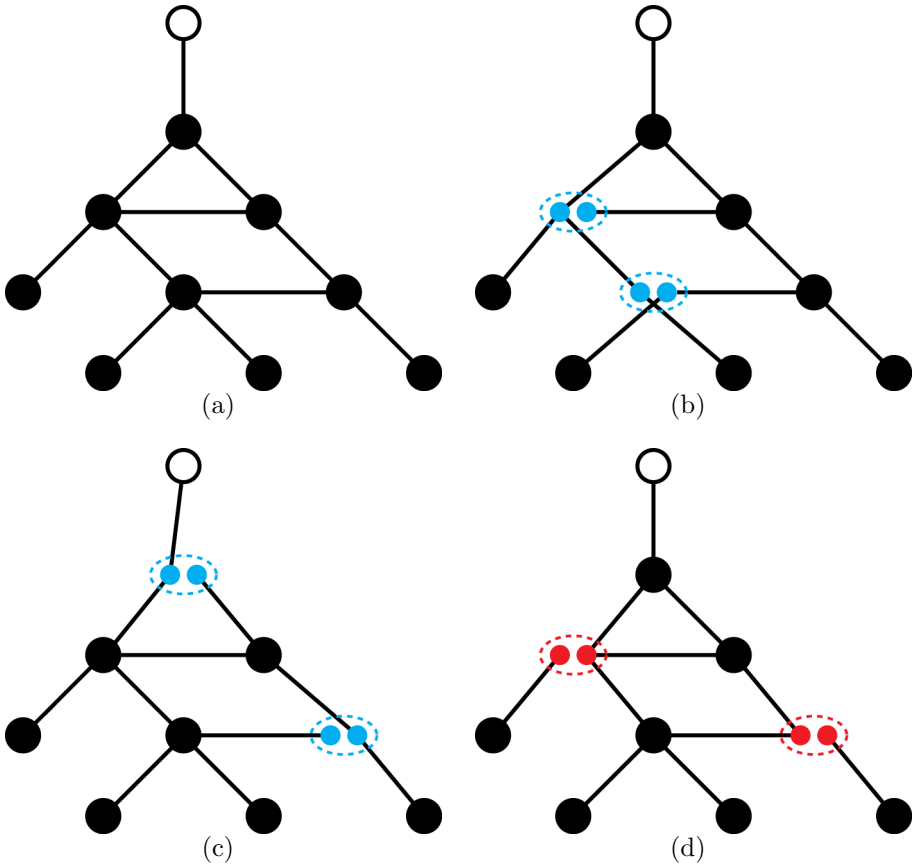


FIGURE 6.1: **Graph multiplicities:** (This Figure is best viewed in color.) Every possible tree that projects down to a planar graph corresponds to a choice of where and how to add the extra vertices such that the tree is connected and has no circuits. **(a)** A planar graph. **(b)** A G-tree. The vertices with multiplicity greater than one are highlighted in light blue. The dotted ovals indicate that the vertices inside of them project to the same location on the plane. **(c)** A different G-tree corresponding to a different choice of vertex multiplicities. **(d)** An invalid graph. The multiplicity choices highlighted in red do not correspond to a tree.

7

Prior Model for Arborescences

The previous chapters discussed the geometry and combinatorics of arborescences and their projections. We now introduce ways to quantify the quality of an estimate of a directed tree inferred from a given image graph. Specifically, in this chapter we introduce a generative, parametric model which defines a probability measure for a wide class of arborescences. In the following chapter, we show that probabilities can also be associated to flow orientations (Chapter 8), and examine the computational complexity of computing the most likely directed tree from a given image graph (Chapter 9).

As noted in Chapter 2, the modeling of tree growth is an active research area in many fields, and models exist for the growth patterns exhibited by a number of different types of trees. Taken as a whole, this work suggests that the forces that guide specific growth patterns are often complex, multi-scale, and interdependent. Overall, the shape of a tree depends on a myriad of global and local interactions that determine when a given branch will spawn new branches, how many children it will have, and in what directions they will grow.

In this paper, we describe the growth of an arborescence generatively as a stochas-

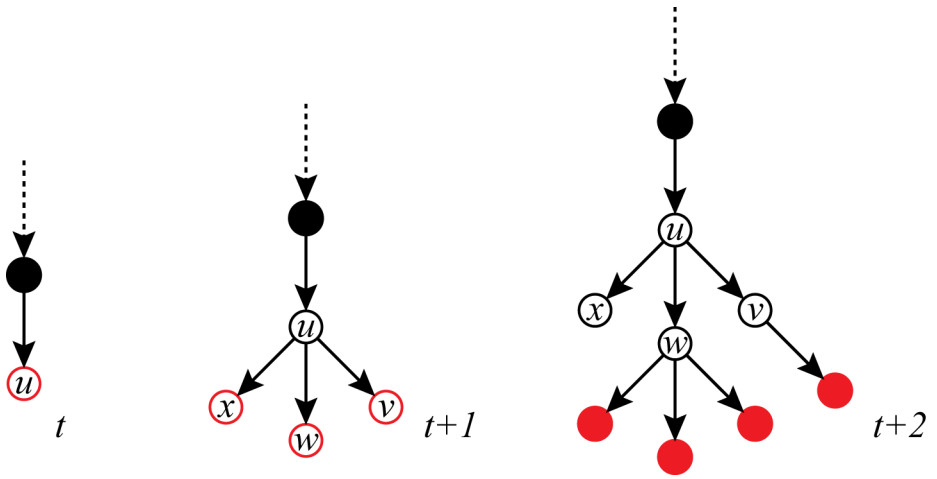


FIGURE 7.1: **Arborescence growth:** At each step, the frontier $F_A(t)$ (highlighted in red) is updated. At depth t, a new vertex u is added to both $F_A(t)$ and $V_A(t)$. At depth $t+1$, u is removed from $F_A(t)$ and its offspring $\{v, w, x\}$ are added to the frontier. At depth $t+2$, the three vertices are removed from the frontier; v is succeeded by a single child and w spawned three children, while x had none.

tic, discrete, spatial Markov branching process that evolves over time. The resulting arborescence consists of a set of a piecewise-linear branches. In our model, an arborescence can only grow by extending its leaf branches that are still "active". Since we do not allow internal branches to increase in length, our model does not describe all actual growth processes in nature, but is rather to be seen as an abstract, generative model of the final shape of an arborescence. Although our model is simple, the experiments in Chapter 12 show that it adequately captures the morphology of a number of different types of trees, including retinal vessels and plant roots. We will first present our model's branching behavior and then detail how we model the geometry of an arborescence.

7.1 Branching behavior

The *frontier* $F(t)$ of a growing arborescence $A(t) = (V_A(t), E_A(t), r_A)$ with embedding $\eta(A(t))$ is the set of $m(t)$ vertices

$$F(t) = \{\mathbf{v}_1, \ldots, \mathbf{v}_{m(t)}\} \subset V_A(t)$$

in $V_A(t)$ that are leaves and are still growing. The index $t \in \mathbb{N}_0$ indicates the tree-depth of the leaves. At depth $t = 0$, the arborescence is a single point—its root r_A, embedded at the origin of space—and $F(0) = \{r_A\}$.

At depth $t > 0$, each leaf $\mathbf{v}$ in $F(t)$ spawns a number $c \geqslant 0$ of labeled branch *stubs* with probability $p_s(c)$, where

$$\sum_{c=0}^{\infty} p_s(c) = 1 \, . \tag{7.1}$$

A stub is a short branch segment that connects a new leaf to its parent. If $c > 0$, we assign an ordered label $i \in \{1, 2, \ldots, c\}$ to each of the children of $\mathbf{v}$. For simplicity, we assume that $p_s(c)$ is the same for all t. In our experiments, it is given by a one-inflated Poisson distribution (Lambert, 1992) that behaves as a Poisson distribution with rate λ at all values except 1:

$$p_s(c) = \begin{cases} \frac{1}{1+\alpha}(\text{Pois}\,(c\,;\lambda) + \alpha) & \text{if } c = 1 \\ \frac{1}{1+\alpha}\,\text{Pois}\,(c\,;\lambda) & \text{otherwise.} \end{cases}$$

In many physical trees, $\alpha >> \text{Pois}\,(1\,;\lambda)$ to account for the fact that extending a branch is much more common than stopping or splitting into two or more branches.

The vertex $\mathbf{v}$ is then removed from the frontier and the leaves of any of the stubs it has spawned are added to $F(t+1)$ (and consequently $V_A(t+1)$). The number $m(t)$ of leaves in the frontier $F(t)$ follows a Galton-Watson process (Watson and Galton, 1875) and is given by the recurrence formula:

$$m(t+1) = \sum_{j=1}^{m(t)} c(t, j),$$

where the random variable $c(t, j)$ denotes the number of stubs of leaf $\mathbf{v}_j \in F(t)$ and had distribution p_s.

7.2 Geometric modeling

To geometrically model stub growth, every new leaf $\mathbf{v}$ in the frontier $F(t)$ records information about its parent $\pi(\mathbf{v}) = \mathbf{u}$ and about its *growth increment*, a three-dimensional random variable $\delta\mathbf{v} = \mathbf{v} - \pi(\mathbf{v})$ that depends on the following factors:

1. The stub's growth inertia
2. The preferred branching angle
3. The force field(s) surrounding $\mathbf{u}$.

Factors 2 and 3 are dependent on the type of arborescence being modeled. Growth inertia refers to the tendency of a branch to continue to grow in a steady direction. A preferred branching angle captures the tendency of angles between a parent and its (multiple) child branches to take on similar patterns for a given type of arborescence and number of children. The force field encapsulates relevant environmental forces, such as a gravitational or electromagnetic field or the density of the growth medium, that affect growth.

These terms are combined as follows. Let $\mathbf{v}_i$ be the i-th labeled child of vertex $\mathbf{u}$. Then its location is given by:

$$\begin{aligned} \mathbf{v}_i &= \mathbf{u} + \delta\mathbf{v}_i, \\ \delta\mathbf{v}_i &\sim \mathrm{VF}(\hat{\boldsymbol{\mu}}_i, \kappa, \gamma) = \gamma \left(\frac{\kappa}{2\pi(e^{\kappa} - e^{-\kappa})} e^{(\kappa \hat{\boldsymbol{\mu}}_i^{\top} \delta\mathbf{v}_i)} \right) \end{aligned} \tag{7.2}$$

In these expressions, $\delta\mathbf{v}_i$ is a vector originating at $\mathbf{u}$ that determines the position of $\mathbf{v}_i$. Its direction is drawn from a von Mises-Fisher distribution (Khatri and Mardia, 1977) with concentration κ and its length is scaled by a parameter $\gamma > 0$, which can be either a constant or drawn from some distribution. The expected direction of growth $\hat{\boldsymbol{\mu}}_i = \frac{\boldsymbol{\mu}_i}{\|\boldsymbol{\mu}_i\|}$ is given by a vector sum that incorporates the three factors

outlined above:

$$\boldsymbol{\mu}_i = \mathbf{s}(\rho_c(i), \phi_c(i), z_c(i)) + \mathbf{f}(\mathbf{u}), \tag{7.3}$$

where c is the number of children of $\mathbf{u}$. Furthermore,

- $\mathbf{s}(\rho_c(i), \phi_c(i), z_c(i))$ is the expected direction of growth of the i-th child of $\mathbf{u}$. We parameterize $\mathbf{s}$ using cylindrical coordinates with origin $\mathbf{u}$ and cylindrical axis along $\delta\mathbf{u}$.[1] $\rho_c(i)$ and $z_c(i)$ are the radius and height, respectively, and $\phi_c(i)$ is the azimuth relative to a random reference plane.

- $\mathbf{f}(\mathbf{u})$ is the vector value for the force field that captures environmental forces at that location.

[1] If $\mathbf{u}$ is the root, then $\delta\mathbf{u}$ is chosen uniformly at random.

8

The Probability of an Arborescence

In this chapter, we show how to evaluate the probability of both three-dimensional and projected arborescences under a growth model $M = (p_s, \boldsymbol{\rho}, \boldsymbol{\phi}, \mathbf{z}, \mathbf{f}, \kappa, \gamma)$ defined by a distribution p_s on the number of children per vertex, the expected coordinates $\rho_c(i)$, $\phi_c(i)$ and $z_c(i)$ for the i-th child of c siblings, the force field $\mathbf{f}(\mathbf{v})$, the concentration parameter κ of the random perturbations, and the scale parameter γ. In our experiments, we estimate these various parameters via a training set of arborescences.

8.1 Arborescence probability

The probability $p_M(A)$ of an arborescence A given M is a function of its topology (the number of children per vertex) and geometry (the angles between the stubs of a parent and those of each of its children). Furthermore, our model is local, in the sense that the expected number of children is the same for every vertex $\mathbf{u}$ and their expected locations depend only on $\delta\mathbf{u}$ and the local force field. Thus, we decompose the above probability as follows:

$$p_M(A) = \prod_{\mathbf{u} \in V(A)} p_M(\mathbf{u}|\pi(\mathbf{u}), c) = \prod_{\mathbf{u} \in V(A)} p_a(\mathbf{u}|\pi(\mathbf{u}), c) p_s(c), \tag{8.1}$$

where $\pi(\mathbf{u})$ is the parent of $\mathbf{u}$. Here, p_s is defined in (7.1) and p_a is the probability of the angles between the parent stub and the labeled stubs of each of its c children:

$$p_a(\mathbf{u}|\pi(\mathbf{u}), c) = \begin{cases} \prod_{\mathbf{v}_i \in V_\mathbf{u}} \mathrm{VF}(\delta\mathbf{v}_i; \hat{\boldsymbol{\mu}}_i, \kappa, \gamma) & \text{if } c > 0 \\ 1 & \text{otherwise,} \end{cases} \tag{8.2}$$

where $V_\mathbf{u}$ are the c children of $\mathbf{u}$, and $\delta\mathbf{v}_i$ is the stub corresponding to i-th labeled child. For each set of siblings, the azimuth angles are measured w.r.t the reference plane that maximizes the above joint probability. For convenience, we can equivalently maximize the *log-probability*:

$$\ell_M(A) = \sum_{\mathbf{u} \in V(A)} \log\left(p_M(\mathbf{u}|\pi(\mathbf{u}), c)\right). \tag{8.3}$$

8.2 Projected arborescence probability

Let $A_P = P(A)$ be a projected arborescence for which we know (or have estimated) its original topology, but not its 3D geometry. Since there are uncountably infinite arborescences that could have projected to A_P, we cannot estimate (8.1) for A_P directly. Instead, we approximate the above probability as follows:

$$p_M(A_P) \approx \prod_{\mathbf{u}_P \in V(A_P)} p_a(\mathbf{u}_P|\pi(\mathbf{u}_P), c) p_s(c), \tag{8.4}$$

where $\mathbf{u}_P$ is the projection of $\mathbf{u}$. The branching probability p_s remains the same after projection, but we cannot evaluate p_a directly. In addition to the lack of 3D information, it is not possible, in general, to recover the original labels of a group of siblings from their 2D positions. We thus approximate p_a by estimating the expected direction of growth of each stub and then approximating the force field.

8.2.1 Direction of growth approximation

We first define a *branching template* $B(c) = \{\delta\tilde{\mathbf{u}}, \delta\tilde{\mathbf{v}}_1, \delta\tilde{\mathbf{v}}_2, \ldots, \delta\tilde{\mathbf{v}}_c\}$ to be the set of labeled, expected 3D stubs for any vertex with c children, ignoring the force field

and any random perturbations; that is, $\delta\tilde{\mathbf{v}}_i = \mathbf{s}(\rho_c(i), \phi_c(i), z_c(i))$. Note that B is invariant under rigid transformations.

Now, let $C(\mathbf{u}_P) = \{\delta\mathbf{u}_P, \delta\mathbf{v}_P(1), \delta\mathbf{v}_P(2), \ldots, \delta\mathbf{v}_P(c)\}$ be the set of projected 2D stubs incident to $\mathbf{u}_P$ ordered clockwise starting from the parent stub $\delta\mathbf{u}_P$. We then seek the projection of B that best approximates C.

More concretely, let $\mathbf{B}_P$ be the set of possible projections of B such that $P(\delta\tilde{\mathbf{u}}) = \delta\mathbf{u}_P$. We seek the projection $B_P^\star$ that minimizes the sum of the 2D angles between the expected and actual stubs:

$$B_P^\star = \underset{\mathbf{B}_P}{\operatorname{argmin}} \left[\min_{S_c} \left(\sum_{i=1}^{c} \arccos(\sigma_i(\delta\mathbf{v}_P)^\top P(\delta\tilde{\mathbf{v}}_i)) \right) \right] \tag{8.5}$$

where S_c is the set of all possible ordered label assignments to the stubs in $C(\mathbf{u}_P)$ and $\sigma_i(\delta\mathbf{v}_P)$ is the stub that is assigned the i-th label in permutation σ.

We estimate (8.5) via a two-step search. We first project $\delta\tilde{\mathbf{u}}$ to $\mathbf{u}_P$ at a finite set of angles in $[0, \pi)$ that define the slant of $\delta\tilde{\mathbf{u}}$ w.r.t. the plane of projection. For each slant, we then rotate the child stubs in B by a set of angles in $[0, 2\pi)$ to find the projection for that slant that minimizes the above sum. Finally, we define $\kappa_{\mathbf{s}}$ to be a concentration parameter given by the variance of the possible angles between expected and actual *projected* vectors.

8.2.2 Force field approximation

Let $\mathbf{f}_P = P(\mathbf{f})$ be the projection of the force field $\mathbf{f}$ onto the plane. At every point $\mathbf{x} \in \mathbb{R}^2$, let the 2D vector $\boldsymbol{\mu}_{\mathbf{f}_P}(\mathbf{x})$ be the mean direction of the projections of the vectors whose origins lie along $\mathbf{x}$'s line of projection and let $\kappa_{\mathbf{f}_P}(\mathbf{x})$ be the concentration parameter given by the variance of their angles. In our experiments, we estimate both parameters by sampling along the line of projection.

8.2.3 Angle probability approximation

We approximate p_a for a set of siblings via a set of von Mises distributions with mean directions $\boldsymbol{\mu}_i = P(\delta\tilde{\mathbf{v}}_i) + \boldsymbol{\mu}_{\mathbf{f}_P}(\mathbf{u}_P)$ and concentration $\kappa_P = \kappa_{\mathbf{s}} + \kappa_{\mathbf{f}_P}(\mathbf{u}_P)$:

$$p_a(\mathbf{u}_P) = \begin{cases} \prod_{\mathbf{v}_P(i) \in V_{\mathbf{u}_P}} \mathrm{VM}(\delta\mathbf{v}_P(i); \hat{\boldsymbol{\mu}}_i, \kappa_P) & \text{if } c > 0 \\ 1 & \text{otherwise,} \end{cases} \tag{8.6}$$

where $\mathbf{v}_P(i)$ has been assigned the i-th label. The von Mises distribution is simply the von Mises-Fisher distribution restricted to the unit circle:

$$\mathrm{VM}(\mathbf{v}; \boldsymbol{\mu}, \kappa) = \frac{e^{(\kappa \boldsymbol{\mu}^\top \mathbf{v})}}{2\pi J_0(\kappa)}, \tag{8.7}$$

where $\mathbf{v}$ and $\boldsymbol{\mu}$ are unit length vectors and J_0 indicates the modified Bessel function of order zero. The von Mises distribution is the circular analogue of the Gaussian distribution, with $\boldsymbol{\mu}$ and κ as the mean and precision, respectively.

9

Tree Estimation Complexity

In the previous two chapters we presented a parametric arborescence growth model and showed how to estimate the probability of a projected arborescence. In this chapter, we will analyze the computational complexity of estimating the most likely tree for a restricted class of *local* tree probability models–that includes our growth model–in which the probability of a tree is the product of the probability of each set of incident edges at each vertex.

First, we will show that we can estimate the probability of a flow-dag in linear time for any model in this class; however, we will then show that determining the optimal tree for this class of models is NP-hard for undirected graphs.

9.1 Tree probability

Let $T = (V_T, E_T, r_T)$ be a directed tree rooted at r_T. In this work, we are interested in a class of probability models that decompose the probability of a tree into the probability of each of its vertices. More specifically, we will restrict ourselves to

models in which the probability of each vertex is a function of its incident edges:

$$\ell(T) = \sum_{v \in V_T} \ell(v | E_T(v), \theta) \, , \tag{9.1}$$

where $\ell(T)$ is the log-probability of the directed tree, $E_T(v)$ is the set of edges in T which are incident to v, and θ are any additional model parameters, such as the force field at v's location. In other words, the probability of a vertex can be fully specified by the properties (such as location or number) of its single incoming edge and its outgoing edges, if any. We will refer to models that satisfy (9.1) as local models. It is easy to see that (8.4) satisfies (9.1), so our growth model is local.

Here, we assume that $\ell(v) = O(1)$; that is, calculating each vertex log-probability takes constant time. In this case, the overall probability of a tree can be computed in linear time by determining the probability of each of its vertices in turn.

9.2 The probability of an acyclic flow orientation

Given a flow-dag G_d of a graph G, we define its log-probability as

$$\ell(G_d) = \max_{T \in \mathcal{T}(G_d)} \ell(T). \tag{9.2}$$

If $\ell(T)$ is defined in terms of a local model, we can efficiently maximize this probability over the set of G_d-trees consistent with G_d. This is because one can consider each crossing x in G_d, evaluate all of the valid partitions of x, and pick the partition whose vertices maximize (9.1). Since changing the partition of x does not change any of the edges that are incident to it, the choice of a most likely, valid partition for x is local, that is, it affects no vertex other than x, nor does it depend on any other vertices.

Consequently, the maximization in this expression can be performed vertex-by-vertex. Therefore, if a *directed* graph G_d were available, it would be possible to find a G_d-tree $T \in \mathcal{T}(G_d)$ by examining each of the vertices of G_d in turn. Since the

degree of the vertices in G_d is assumed to be bounded, the time complexity at each vertex is constant, and the overall complexity of estimating the most likely directed tree $T \in \mathcal{T}(G_d)$ is linear in the number of vertices in the graph G_d.

Unfortunately, if only the *undirected* graph G is available, finding an optimal directed tree $T \in \mathcal{T}(G)$ given a local model is NP-hard. We prove this in the following section by reducing the classic minimum vertex cover problem (Garey and Johnson, 1979) to the problem of estimating a tree from an undirected graph.

9.3 Optimal estimation from undirected graphs is NP-hard

We show that finding an optimal G-tree $T \in \mathcal{T}(G)$ is NP-hard by reducing the minimum vertex cover (MVC, (Garey and Johnson, 1979)) problem to a directed-tree estimation (DTE) problem. MVC is NP-complete even for graphs with bounded degree (Austrin et al., 2009).

A *vertex cover* of a graph $H = (V_H, E_H)$ is a subset $C \subseteq V_H$ such that every edge of E_H is incident to at least one vertex in C. The bounded-degree version of MVC is defined as follows: Given a connected graph H with bounded degree, find a vertex cover of smallest possible size.

Let G be an undirected graph with root r_G. DTE is defined as follows: Find

$$T^\star = \operatorname*{argmax}_{T \in \mathcal{T}(G)} \ell(T)$$

where $\ell(T)$ is defined in terms of a local model (see (9.1)), and $\mathcal{T}(G)$ is the set of directed trees–for which every edge is directed away from the root–that project to G and $P(r_T) = r_G$. In this reduction, we assume, w.l.o.g., that $\theta = \varnothing$.

In order to construct a polynomial-time transformation from MVC to DTE, we first give a way to interpret a graph orientation as a vertex cover. Let H_d be an orientation of the undirected graph H and define

$$C = \{v \in V_H \mid \deg^+_{H_d}(v) > 0\} \tag{9.3}$$

where $\deg^+_{H_d}(v)$ is the out-degree of v in H_d. The set C is a vertex cover of H, because every edge starts at some vertex, and that vertex is in C by construction.

We now show a way to use DTE to find a minimum vertex cover for a given graph H, thereby reducing MVC to DTE. Specifically, we first modify H into a new graph G so that every orientation in H is a flow orientation in G. We then define a log-probability function of the form (9.1) on G such that increasing the log-probability is equivalent to reducing the number of vertices in the set C defined above. Thus, a solution to DTE for G yields a minimum cover for H.

Let $G = (V_G, E_G, r_G)$ be the rooted graph obtained from H by adding a new root vertex r_G and new edges to connect r_G to every vertex in V_H, and let $\mathcal{T}(G)$ be the set of directed trees consistent with G.

We will now define a local model over $\mathcal{T}(G)$. Let $T \in \mathcal{T}(G)$ be a directed tree consistent with G. Then, we define the log-probability of each of its vertices as follows:

$$\ell(v|E_T(v)) = \begin{cases} -1 & \text{if } \deg^+_T(v) > 0 \quad \text{and} \quad v \neq r_T \\ 0 & \text{otherwise.} \end{cases} \tag{9.4}$$

By definition of $\mathcal{T}(G)$, the root r_T of T has in-degree zero. In short, each vertex, other than the root, that has at least one child makes the tree less likely.

Every directed tree in $\mathcal{T}(G)$ projects to a unique flow orientation of G rooted at r_G. Conversely, given a flow orientation G_d of G rooted at r_G, we can obtain the most likely tree consistent with it by maximizing (9.4) at each node as follows.

We divide the nodes of G_d into four types: the root r_G, non-root nodes with out-degree zero, non-root nodes with positive out-degree and in-degree one, and non-root nodes with positive out-degree and in-degree greater than one. Nodes in the last category (and possibly some in the second) are crossings.

The maximum possible log-probability achievable at each node type is as follows. The root has in-degree zero because we do not allow roots to be crossings (see

Section 4.1), and we show in Section 5.1 that under this assumption the root cannot have positive in-degree. Because of this, the root has a log-likelihood of zero by (9.4). Non-root nodes with out-degree zero can only be partitioned into tree leaves, and these have likelihood zero by (9.4) as well. Non-root nodes with positive out-degree and in-degree one are not split, and they get log-likelihood -1 by (9.4). Finally, any crossing v with positive out-degree in G_d must be split into a number of nodes equal to its in-degree in G_d. Since the out-degree of v is positive, its outgoing edges must be assigned to at least one of the nodes in the partition. Each such assignment results into a log-probability term of -1 by (9.4). As a consequence, the overall log-probability at this node is at most -1, because log-probabilities add up according to (9.1). This upper bound on the log-probability can be made tight by assigning all the outgoing edges of v to the same node of the partition.

To summarize, the maximum possible log-probability at each node of G_d given (9.4) is given by:

$$\ell(v|E_{G_d}(v)) = \begin{cases} -1 & \text{if } \deg^+_{G_d}(v) > 0 \quad \text{and} \quad v \neq r_G \\ 0 & \text{otherwise.} \end{cases}$$

Every possible orientation H_d of H corresponds to a unique flow orientation G_d of G rooted at r_G, because every vertex in V_H is reachable from r_G through at least one directed path in G_d. Furthermore, each vertex in V_H which has an outgoing edge in E_{H_d} (and consequently in E_{G_d}) makes the flow orientation less likely. Therefore, a flow orientation of G with maximum probability has a minimal number of vertices in V_H with non-zero out-degree in E_{H_d}. The set C resulting from the interpretation (9.3) of H_d is a minimum vertex cover of H, so MVC reduces to DTE.

The above result holds even if the orientations of G are restricted to be flow-dags because, for every minimum vertex cover C, there always exists at least one flow orientation of G, rooted at r_G, consistent with C that is acyclic. To construct a

flow-dag given C, first assign a topological ordering to the vertices in G such that:

$$r_G < v, \quad \forall v \in V_H$$

$$v < u, \quad \forall v \in C, u \in V_H \setminus C.$$

In other words, all the vertices in V_H that are part of the vertex cover of H come before those not in the cover. Then, orient every edge in G according to the topological ordering. The resulting flow orientation is acyclic and a vertex in V_H has at least one outgoing edge in E_{H_d} if and only if it is part of C.

10

Optimal Tree Search

Since finding an optimal G-tree is NP-hard, we resort to approximate methods. Specifically, this section present a two-step approach for estimating a highly likely G-tree from an undirected graph G. The algorithm in the first step finds an approximate solution greedily, and the one in the second step improves this solution via a heuristic search in the space of possible G-trees.

10.1 Greedy Directed Tree Search

Computing a G-tree from an undirected, rooted graph $G = (V, E, r)$ involves the two choices mentioned in Chapter 6: assign a multiplicity (or equivalently an in-degree) to each vertex v of G other than r—which defines an orientation over G—and construct a valid partition of v, as defined in Chapter 4. If the resulting orientation is a flow-dag, any choice of valid partitions will yield a G-tree, so we restrict our search to acyclic orientations.

To estimate a high-probability G-tree, we first obtain a flow-dag G_d by completing a shortest-path spanning tree of G. We then apply a valid partition to each vertex with in-degree $\geqslant 2$ to convert G_d into a directed tree.

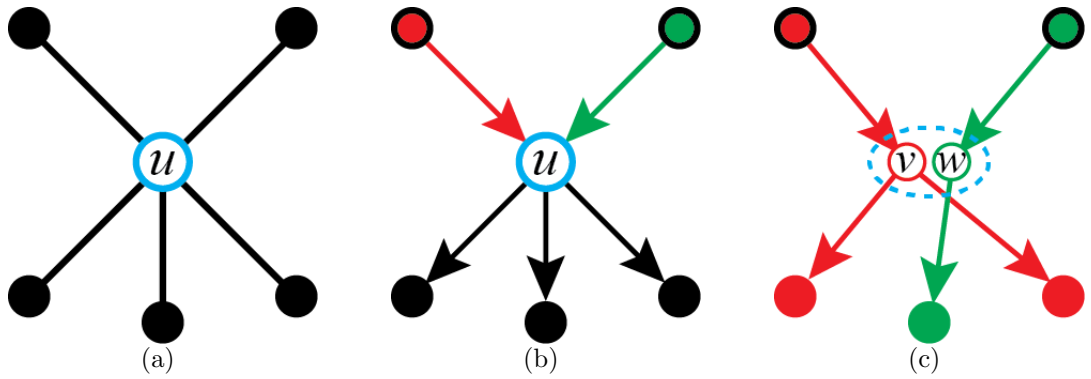


FIGURE 10.1: **Valid partition formation:** (This Figure is best viewed in color.) **(a)** A vertex u and its neighborhood. In **(b)**, the edges incident to u are assigned orientations. Since u has more than one incoming edge (highlighted in red and green), it must be partitioned. In **(c)**, u is split into two vertices v (in red) and w (in green) such that each vertex has a single parent. The dotted oval indicates that v and w share the same location on the plane. Each of the outgoing edges of u is then assigned to one of the new vertices.

In some more detail, a shortest-path tree D rooted at r is a spanning tree of G such that for any vertex v, the shortest distance $\text{dist}\,(r, v)$ between r and v is the same in G and in D (Khuller et al., 1995). When the cost of each edge is the Euclidean distance between its two vertices, then D approximates the arrival time of a flow sent from r to every other vertex in the graph. A (not necessarily unique) shortest-path tree D can be efficiently estimated in time $O(|E| + |V| \log(|V|))$ using Dijkstra's shortest-path algorithm (Dijkstra, 1959). At each iteration, Dijkstra's algorithm extends the shortest-path tree by one vertex using the function

$$v = \texttt{closest\_vertex}(G, V_{\text{visit}})$$

which find the closest unvisited vertex v to the set of visited vertices V_{visit}.

A shortest-path tree induces a topological ordering of the vertices of G in which $u < v$ if and only if $\text{dist}\,(r, u) \leqslant \text{dist}\,(r, v)$.[1] Given D, we construct a flow-dag G_d by orienting each edge in G so that it obeys a topological ordering defined by D. It is easy to show that the resulting directed graph G_d is a flow-dag: since it contains the

[1] If the two vertices are the same distance away from the root, they are ordered arbitrarily.

spanning directed tree D as a sub-graph, it is a flow orientation and, since its edges obey a topological ordering, it is a dag.

As noted in Section 9.2, given a flow-dag, we can determine a most likely directed tree in linear time by partitioning each crossing of G_d optimally and so that every vertex has a single parent. For a given crossing vertex $v \in V(G_d)$, the optimal value of $\ell(v)$ is found by a function

$$(\ell, P) = \texttt{partition}(v, G_d, M)$$

that uses model M to evaluate in turn the likelihood of all valid partitions of v that can be obtained given the directions in G_d of the edges incident to v, and picks the partition with highest likelihood. The function `partition` also returns a structure

$$P = (P.v, P.V, P.E, P.J)$$

that describes the surgery necessary to implement the partition. This includes the vertex $P.v$ to be replaced, the set $P.V$ of new vertices that replace $P.v$, the set $P.E$ of oriented edges that are incident to $P.v$, and a set $P.J$ of indices that for each edge e in $P.E$ tells what vertex of $P.V$ the edge e connects to. See Figure 10.1.

A procedure

$$G = \texttt{transform}(G, P)$$

then transforms the graph G to implement the optimal partition P: It replaces $P.v$ with the elements of $P.V$ and replaces the $P.v$ endpoint of each edge in $P.E$ with the vertex in $P.V$ indicated by $P.J$. Algorithm 1 summarizes our greedy search approach.

10.2 Heuristic Directed Tree Search

The algorithm presented in the previous subsection finds a highly likely G-tree from graph G, but does so greedily. We now introduce a heuristic search algorithm that

Algorithm 1: Greedy search for a likely G-tree

Input: Undirected, rooted planar graph $G = (V, E, r)$, tree growth model M
Output: A high-likelihood directed tree T

```
/* Obtain a shortest path tree                                    */
```
$V_{\text{visit}} = \{r\}$
while $|V_{visit}| < |V|$ **do**
    $v =$`closest_vertex`(G, V_{visit})
    $V_{\text{visit}} = V_{\text{visit}} \cup \{v\}$
    $d[v] = \text{dist}(r, v)$

```
/* Determine a corresponding flow-dag                             */
```
$E_d = \varnothing$
$G_d = (V, E_d, r)$
foreach $e = (u, v) \in E$ **do**
    if $d[u] \leqslant d[v]$ **then**
        $E_d = E_d \cup \{(u, v)\}$
    else
        $E_d = E_d \cup \{(v, u)\}$

```
/* Convert flow-dag to tree                                       */
```
$T = G_d$
foreach $v \in T$ **do**
    if $\deg^-(v) > 1$ **then**
        $(\ell, P) =$ `partition`(v, T, M)
        $T =$ `transform`(T, P)

attempts to improve on the greedy solution by exploring variants of it that may increase its likelihood.

The expression (9.2) in Section 9.2 evaluates the likelihood of every node of the flow-dag meta-graph defined in Section 5.3. To efficiently explore this graph, we make use of a heuristic that encourages moving towards a high-probability *anchor*, defined as follows.

Given a growth model, every possible valid partition of a vertex $v \in V_G$ has an associated probability. Then, let L be the *unconstrained* list of optimal partitions of G, one for each vertex. By "unconstrained" here we mean that we ignore whether vertex partitions are consistent with edge orientations: Recall that a partition at v

Algorithm 2: Heuristic directed tree search

Input: Graph G, Greedy flow-dag G_d, growth model M, max priority queue q, max number of iterations $i_{\max} > 1$
Output: Locally optimal directed tree $T^\star$.

```
L = unconstrained_partitions(G,M)
G_L = anchor(L,M)
ℓ* = ℓ_M(G_d)
q = push(G_d, h(G_d, G_L))
for i ← 1 to i_max do
    G_d = pop(q)
    if ℓ* < ℓ_M(G_d) then
        ℓ* = ℓ_M(G_d)
        G*_d = G_d
    foreach e ∈ E(G_d) do
        if is_crossing(e,G_d) then
            G'_d = flip_edge(e,G_d)
            if not_visited(G'_d) & is_dag(G'_d) then
                q = push(G'_d, h(G'_d, G_L))
T* = G*_d
foreach v ∈ T* do
    if deg⁻(v) > 1 then
        (ℓ, P) = partition(v, T*, M)
        T* = transform(T*, P)
```

assigns an orientation to each of the edges incident to v.

However, an edge $[u, v]$ may be assigned different directions by the two partitions at u and v, so in general L does not define a unique orientation of G. We convert L to an orientation by assigning to each edge with two directions its *preferred direction*, which we define as the direction of flow that is more likely given the force field at that location.

More concretely, let $\mathbf{e}_u = (u, v)$ and $\mathbf{e}_v = (v, u)$ be the vectors that correspond to the two possible orientations of the edge e. Then, we define the probability of each

orientation using two von Mises distributions:

$$p(\mathbf{e}_v) = \frac{\mathrm{VM}(\mathbf{e}_v; \boldsymbol{\mu}_\mathbf{f}(v), \kappa_\mathbf{f}(v))}{\mathrm{VM}(\mathbf{e}_v; \boldsymbol{\mu}_\mathbf{f}(v), \kappa_\mathbf{f}(v)) + \mathrm{VM}(\mathbf{e}_u; \boldsymbol{\mu}_\mathbf{f}(u), \kappa_\mathbf{f}(u))}$$

and likewise for $p(\mathbf{e}_u)$. Here, $\boldsymbol{\mu}_\mathbf{f}$ and $\kappa_\mathbf{f}$ are defined as in (8.6). We refer to the orientation with the higher probability, if any, as the preferred orientation of e.

Thus, let the anchor G_L be an orientation of G in which we assign to every edge either the orientation defined by L—if that orientation is unique—or its preferred direction—if L induces conflicting orientations on the edge.[2] If every vertex has a unique most likely partition and all edges have a unique preferred direction, then G_L is also unique.

In general, G_L will not be a flow orientation, and is therefore not a node on the flow-dag meta-graph. However, the probability of L provides an upper bound on the probability of the most likely G-tree and, since G_L is derived from L, any flow-dag that is a few edge-flips away from G_L it likely to be good. Because of this, our search algorithm encourages exploring flow-dags that are near the anchor G_L. Figure 10.2 provides a schematic of the relationship between the unconstrained list, the anchor graph and the meta-graph.

To achieve this goal, the *heuristic value* of exploring a flow-dag G'_d from its neighbor G_d on the flow-dag meta-graph is defined as follows:

$$h(G_d \to G'_d) = \lambda \ell_M(G'_d) + (1 - \lambda) \log\left(1 - \|G'_d, G_L\|_f\right)$$

where λ is a parameter between 0 and 1 and $\|\|_f$ is the number of flips between the two orientations divided by the number of edges in $E(G'_d)$ (note that $|E(G'_d)| = |E(G_L)|$). The first term is the actual likelihood of G'_d, while the second term estimates its nearness to the anchor.

Using this heuristic in a best-first search of the meta-graph leads to Algorithm 2, which starts from the graph orientation corresponding to the directed tree found by

[2] If the two directions are equally probable, we select one arbitrarily.

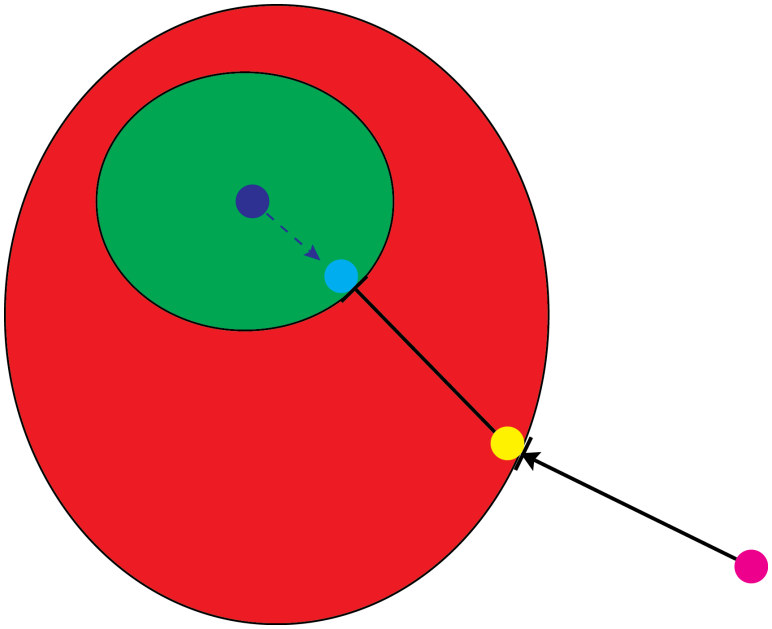


FIGURE 10.2: **Anchor graph:** Schematic representation of the relationship between the unconstrained list, the anchor graph and the set of valid trees. The meta-graph of all possible edge orientations is shown as a red oval; the smaller, green oval represents the connected subgraph corresponding to the meta-graph of all possible flow-dags. The unconstrained list, which may not be a proper orientation, is shown as the magenta dot. The anchor graph (in yellow) is expected to be similar to the unconstrained solution and hence also have high probability. We encourage moving the greedy solution (dark blue) towards a solution (light blue) which is more similar to the anchor graph.

Algorithm 1 and uses a priority queue to implement best-first traversal of the flow-dag meta-graph. By exploring flow orientations in a best-first fashion, this Algorithm is more likely to quickly find locally optimal orientations.

In the next chapter, we will present several ways to quantify the differences between two trees consistent with the same graph, which we will then use in Chapter 12 to quantify how well our estimation algorithms approximate the original tree.

11

Tree Similarity

In the previous chapters, we formalized the problem of estimating the most likely tree given a planar graph and presented two heuristic algorithms to estimate this solution. In order to quantify how well our algorithms approximate the original tree, here we define six different similarity scores between two trees. Each score captures a different property of each tree's topology. We will then make use of these scores in our experiments in the following chapter.

Let T and T' be two different directed trees consistent with the planar graph G. In order to assess the quality of our tree estimation, we need to quantify how similar T and T' are to each other. By construction, these two trees will share the same embedded edges; they will only differ in how these edges are connected to each other. Thus, here we define the similarity between two trees in terms of their edge topology.

However, there is no single metric that can capture all the ways in which two trees differ because trees have properties at multiple scales. Small changes in local topology can produce large changes in the overall structure of the tree. For instance, changing the parent of an edge implicitly changes the ancestor line of all the descendents of

that edge. In other words, changing how two edges are connected affects all the paths from the root that include those edges.

Therefore, here we quantify the similarity between two trees using six similarity scores which measure differences in edge connectivity at multiple scales. Each score captures a different property that may vary between the two trees. Together, they provide a thorough assessment of well as estimated tree approximates the correct solution. We will first define two *local* or edge-based similarities and then define four *global* or path-based similarities. Our global scores are further subdivided into *absolute* and *relative* similarities, as defined below.

11.1 Local similarities

Our two local or edge-based similarities capture differences in local edge connectivity. More concretely, let T and T' be two directed trees consistent with G and let $e_i = (\mathbf{u}, \mathbf{v}) \in E(G)$ be the i-th edge of G given some arbitrary ordering. Then, let $m = |E(G)|$ and let $\mathbf{d}$ be an m-dimensional vector such that:

$$\mathbf{d}(i) = \begin{cases} \|\mathbf{u} - \mathbf{v}\|_2 & \text{if } e_i \text{ has at least one neighbor that is part of a circuit} \\ 0 & \text{otherwise.} \end{cases}$$

We exclude edges with no circuit neighbors because their respective parents will be the same for all solutions. Now, let $\mathbf{o}_p$ and $\mathbf{o}_f$ be two m-dimensional binary vectors such that $\mathbf{o}_p(i) = 1$ if and only if this edge has a different parent edge in T than it does in T' and $\mathbf{o}_f(i) = 1$ if and only if the i-th edge is oriented differently in the two directed trees. Then, the normalized, weighted *parent similarity* s_p and *flow similarity* s_f are given by:

$$s_p = 1 - \frac{\mathbf{o}_p^\top \mathbf{d}}{\|\mathbf{d}\|_1} \quad \text{and} \quad s_f = 1 - \frac{\mathbf{o}_f^\top \mathbf{d}}{\|\mathbf{d}\|_1}.$$

Thus, s_p measures the percentage of edges which have different parents in the two trees, while s_f measures the percentage of edges which are oriented differently in the

corresponding flow orientations.

11.2 Global similarities

As noted above, trees have properties at multiple scales. To capture larger differences in the overall connectivity of two trees, we make use of four similarity scores that capture how the sets of paths from the root differ in the two trees. Comparing the sets of paths in each tree gives us a more global view of the trees' topologies since a change at one edge will affect the paths to many other, potentially distant edges. Furthermore, unlike the two local similarities, which weigh all errors equally, path similarities penalize errors near the root more than errors in the periphery.

Our four path similarities are subdivided into absolute and relative scores. In the absolute scores, two paths are considered different if their edge sets are not identical; in the relative scores we instead consider the percentage of edges which are the same in both paths.

One potential issue with considering all the paths from the root is that these paths will be highly redundant because paths between different edges can have considerable overlap. To reduce this redundancy, we also consider two path similarities only in terms of the leaves of the initial graph G, as defined below.

11.2.1 Absolute similarities

Let $E_{r,e_i}(T)$ and $E_{r,e_i}(T')$ be the respective paths from the root to the edge e_i in the two trees. Then, let $\mathbf{o}_a$ be an m-dimensional binary vector such that $\mathbf{o}_a(i) = 1$ if and only if:

$$E_{r,e_i}(T) = E_{r,e_i}(T).$$

Also, let $\mathbf{l}$ be an m-dimensional binary vector such that $\mathbf{l}(i) = 1$ if and only if one of the vertices incident to e_i is a leaf and let $\mathbf{d}_l = \mathbf{l} \circ \mathbf{d}$, where $\circ$ denotes the Hadamard

or element-wise product between the two vectors. Then, the normalized, weighted *absolute ancestor similarity* s_a and *absolute leaf similarity* s_l are given by:

$$s_a = 1 - \frac{\mathbf{o}_a^\top \mathbf{d}}{\|\mathbf{d}\|_1} \quad \text{and} \quad s_l = 1 - \frac{\mathbf{o}_a^\top \mathbf{d}_l}{\|\mathbf{d}_l\|_1}.$$

Here, s_a measures the percentage of all paths from the root which are identical in the two trees, while s_l is restricted to paths from the root to the leaves of G.

11.2.2 Relative similarities

In addition to the absolute similarities defined above, here we also define two *relative* path similarities that captures what percentage of edges in each path from the root is shared by the two trees. Unlike the local or absolute similarities, however, these relative scores are not symmetric. Nevertheless, they provide a more nuanced assessment of how similar the sets of paths between an estimated and a ground truth tree are. Here, we define these scores for T relative to T'.

As before, let $E_{r,e_i}(T)$ be the subset of edges that form the path from the root to e_i in T. Then, let $S_{r,e_i} \subset E_{r,e_i}(T)$ be the edges that are *shared* in the two trees. Then, we define two vectors that tally the costs of the shared and total edges in the paths from the root:

$$\mathbf{d}_s(i) = \begin{cases} \dfrac{\sum_{e \in S_{r,e_i}} \mathbf{d}(e)}{\sum_{e \in E_{r,e_i}} \mathbf{d}(e)} & \text{if } e_i \text{ has at least one neighbor that is part of a circuit} \\ 0 & \text{otherwise.} \end{cases}$$

Note that $\mathbf{d}_s$ will be different if we instead compare T' to T because the shared edges are compared to the full path, which may differ in the two trees. In addition, let $\mathbf{d}_b = \mathbf{l} \circ \mathbf{d}_s$. Then, the normalized, weighted *relative ancestor similarity* s_r and *relative leaf similarity* s_b are given by:

$$s_r = 1 - \frac{\mathbf{o}_a^\top \mathbf{d}_s}{\|\mathbf{d}_s\|_1} \quad \text{and} \quad s_b = 1 - \frac{\mathbf{o}_a^\top \mathbf{d}_b}{\|\mathbf{d}_b\|_1}.$$

Table 11.1: Tree similarities

Name	**Symbol**	**Description**
Parent similarity	s_p	The edge has the same parent.
Flow similarity	s_f	The edge is oriented in the same direction.
Absolute ancestor similarity	s_a	The path from the root to the edge is identical.
Absolute leaf similarity	s_l	The path from the root to the leaf is identical.
Relative ancestor similarity	s_r	The percentage of the shared path from the root to the edge.
Relative leaf similarity	s_b	The percentage of the shared path from the root to the leaf.

where $\mathbf{d}_s$ is defined with respect to T. Here, s_r measures the weighted percentage of each path from the root which T shares with T', while s_l is restricted to paths from the root to the leaves of G.

Table 11.1 provides a summary of our six similarity scores. In the following chapter, we will apply our two-step tree estimation approach to datasets from several types of trees, including retinal vessels and plant roots and then quantify the quality of our estimation using the scores defined above.

12

Experiments

In this work, we formalized the problem of finding the most likely three-dimensional tree given a two-dimensional projection and presented two algorithms that allow us to explore the space of possible solutions. In order to validate the effectiveness of these algorithms we constructed and analyzed three datasets: a retinal vessel, a rice plant, and a synthetic leafy tree dataset. The first two datasets test how our algorithms perform in two different real world applications, while the third allows us to gain further insight into their strengths and limitations.

For each dataset, we constructed a set of planar graphs and obtained their ground truth trees. We then quantified the similarity between the best trees obtained by our algorithms to the ground truth trees using the scores defined in Chapter 11. We will first describe our experimental methodology and then discuss our results in detail.

12.1 Materials

We obtained three different datasets corresponding to three different kinds of three-dimensional trees: a retinal vessel dataset (15 images), a rice plant dataset (18 image/volume pairs) and a synthetic leafy tree dataset (18 arborescences). These

different datasets allowed us to test our tree estimation algorithms under various scenarios; we now describe each dataset in turn.

12.1.1 Retinal vessel dataset

We constructed a new retinal vessel dataset (WIDE) of 15 high-resolution, wide-field, RGB images using an Optos 200Tx ultra-wide-field device (Optos plc, Dunfermline, Scotland, UK). All images were acquired at the Duke University Medical Center, Durham, NC, USA between August 2010 and October 2012 by Duke Eye Center staff under the supervision of Prof. Priyatham S. Mettu. Each retinal image was taken from a different individual. Each image was captured as an uncompressed TIFF file at the widest setting available for the Optos device (3900×3072 pixels). We then manually cropped away the eyelashes and other non-retinal regions of the image. Afterwards, we downsampled each cropped image by a factor of 2 to obtain the final images (∼900×1400 pixels). Figure 12.1 shows an image before and after cropping, while Figure 12.2 illustrates some additional sample images from this dataset.

12.1.2 Rice plant dataset

We also constructed an 18 image rice root dataset (RICE) by randomly selecting a subset of images from an earlier dataset presented in (Zheng et al., 2011; Iyer-Pascuzzi et al., 2013). In the original dataset, 40 rice plants roots were grown and imaged at the lab of Prof. Philip Benfey at Duke University; each plant was grown in a transparent gel in a separate container. For imaging purposes, each container was placed on top of a turntable, which was programmed to alternate between a small rotation and a stop, long enough for a single image to be acquired. Each plant was imaged at 40 different angles, from which a 3D tomographic model of each plant was obtained using a regularized visual hull algorithm.

To construct our RICE dataset, we first obtained 18 of the 3D model/40 image

(a)

(b)

FIGURE 12.1: **WIDE image acquisition:** (This Figure is best viewed on-screen.) **(a)** We captured each retinal image at the widest setting possible in the Optos 200Tx and **(b)** then manually cropped the image to remove the eyelashes and other non-retinal parts of the image. Both images are at the same scale.

sets corresponding to 13 different plants. Five plants were imaged twice: at 7 and 10 days of growth. For each volume, we randomly selected one of the ∼1300×900 pixel RGB source images. Four sample images from this dataset are shown in Figure 12.3.

12.1.3 Synthetic leafy tree dataset

Finally, we constructed a synthetic leafy tree dataset (SKETCH) of 18 arborescences. Each arborescence was drawn by the present author on the tree modeling software

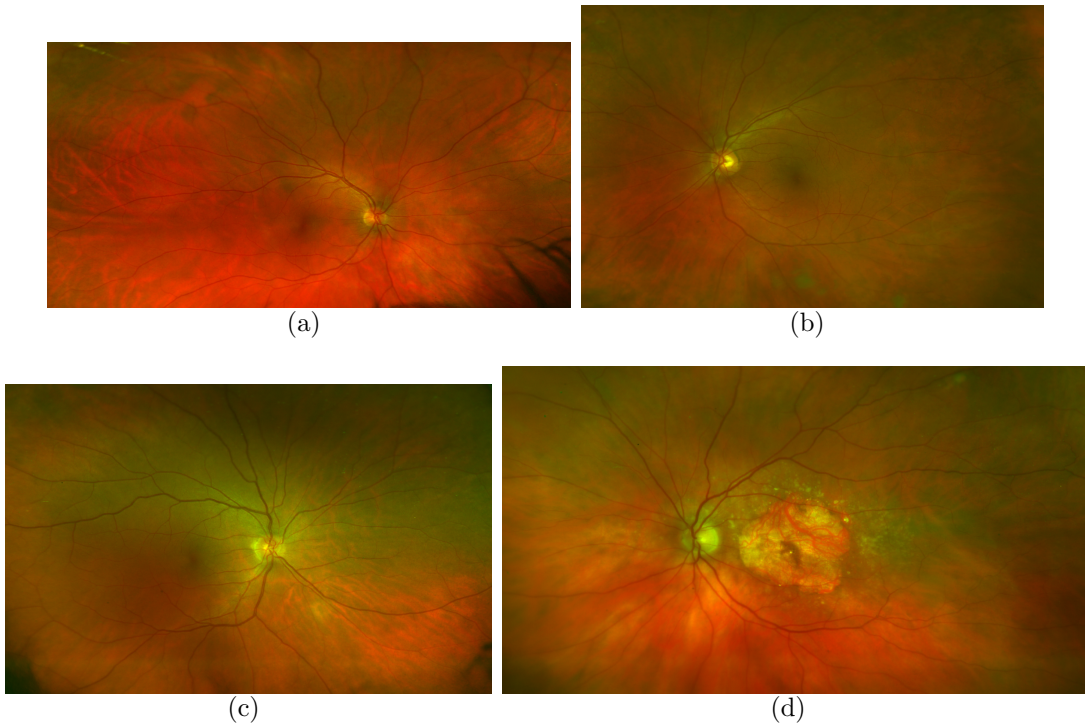


FIGURE 12.2: **WIDE dataset:** (This Figure is best viewed on-screen.) We captured 15 high-resolution images of different retinas using an Optos 200Tx. The various images include left **(a,c)** and right eyes **(b,d)**, as well as pathological eyes **(d)**. All images are shown at the same scale.

developed by Chen et al. (2008) using a Wacom Intuous 3 graphics tablet (Wacom Co. Ltd, Kazo-shi, Saitama, Japan). The sketching software estimates a 3D arborescence given a set of input strokes that represent the tree's branches. Each vertex in the resulting graph is assigned a depth based on the chosen tree template. We used a maple template for all graphs. We then projected each arborescence at five different angles to obtain 90 planar graphs, as detailed in Section 12.2.1.

Our SKETCH dataset is subdivided into three levels of difficulty, each with 6 arborescences. Figure 12.4 illustrates two sample graphs from each of the three categories. The simple arborescences were made using between 20 and 30 strokes, the medium between 40 and 50 and the complex cases between 60 and 70. Note that, regardless of the number of strokes, the number of faces in two planar graphs

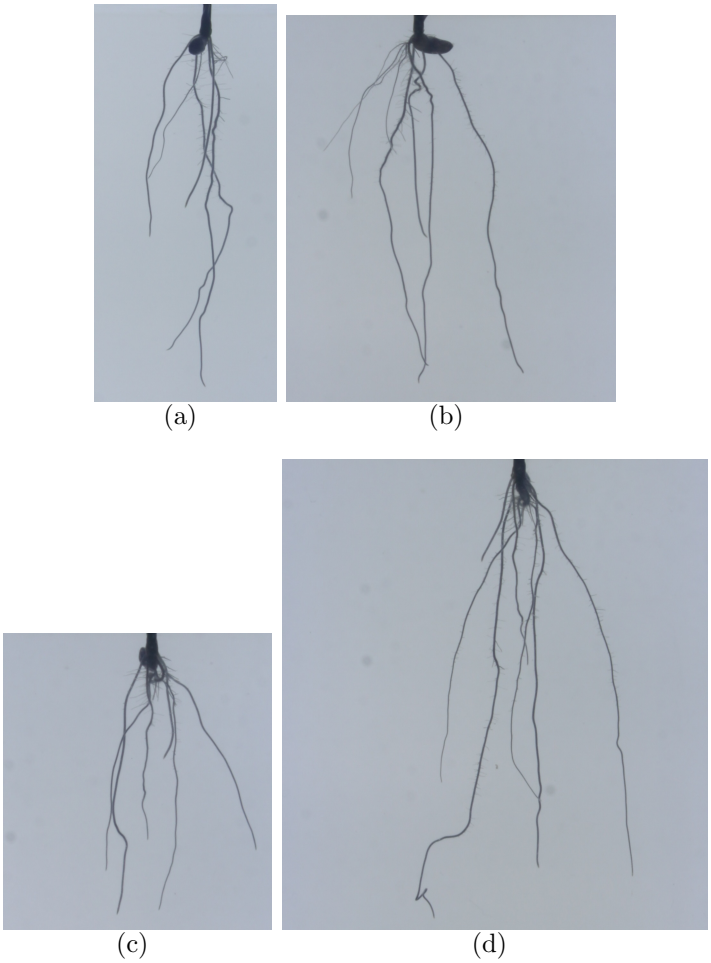


FIGURE 12.3: **RICE dataset:** (This Figure is best viewed on-screen.) Four sample images from our rice plant dataset. **(a)** and **(b)** are different plants, while **(c)** and **(d)** is the same plant imaged at 7 and 10 days of growth, respectively. All images are shown at the same scale

of the same arborescence can vary greatly depending on the angle of projection. For instance, as shown in Figure 12.5, the same arborescence can project to two planar graphs with 7 and 71 faces respectively. Table 12.1 summarizes the statistics on the numbers of strokes and faces for each subset of this dataset.

12.2 Methods

After acquiring the initial images and arborescences, we applied our tree estimation algorithms to each dataset as follows. First, we extracted the planar graphs and

Table 12.1: SKETCH - Projection complexity

Arborescences	Strokes	Faces
All arborescences	45.67 (± 16.18)	37.68 (± 24.37)
Complex subset	64.17 (± 2.48)	56.4 (± 24.13)
Medium subset	46.83 (± 1.47)	38.47 (± 20.37)
Simple subset	26.17 (± 2.32)	18.17 (± 8.27)

obtained the ground truth tree for each graph in each dataset. Then, we applied our greedy search and heuristic search algorithms to each planar graph. We describe each step in more detail below.

12.2.1 Planar graph estimation

For the WIDE and RICE datasets, we obtained each planar graph semi-automatically using the pipeline described in Appendices A to C. After extracting the noisy graph as described in Appendix C, we manually edited each graph using a graph editing software that we developed to correct any errors, such as missing or spurious edges.

For the SKETCH dataset, we first aligned the trunk of every arborescence with the z-axis. We then defined each vertex in the arborescence in terms of a cylindrical coordinate system. We obtained each planar graph by rotating an arborescence by a given angle of rotation ϕ and then collapsing the y-axis. We used five evenly spaced angles ($[0, 1.26, 2.51, 3.77, 5.03]$ in radians) per arborescence to obtain 90 planar graphs in total.

To simulate the limited resolution of an imaging system, we defined a minimum separation radius r of 5 pixels. Any cluster of vertices that lied within a circle of radius r were merged into a single vertex. Then, for every vertex we determined the Euclidean distance d to the closest non-adjacent edge. If $d < r$, we shifted the vertex to lie directly over the edge with probability $\frac{r-d}{r}$; that is, the closer the vertex, the higher the probability that it would be merged.

12.2.2 Ground truth estimation

The SKETCH dataset already includes the ground truth trees. For the WIDE and RICE datasets, we manually obtained the ground truth trees using the aforementioned graph editing software. Our software allows a user to partition a vertex or undo an existing partition. To simplify the above process, we first obtained a non-optimized solution using our heuristic search algorithm and then replaced the invalid partitions with the correct ones. For the RICE dataset, we used the additional images and the 3D volume to determine the correct topology.

For the WIDE dataset, on the other hand, our ground truth trees are based solely on human analysis. Since even human experts differ on their assessment of vessel topology, we quantified the degree of *inter-observer variability* or uncertainty in our ground truth by comparing the trees produced by two human raters (the present author and Prof. Sina Farsiu at the Duke Eye Center) using each of the similarities defined in Chapter 11. We set the first rater's trees as ground truth. The similarity values of the second rater's trees in the test set of the WIDE dataset are listed in Tables 12.3 to 12.5.

12.2.3 Directed tree estimation

We randomly split each dataset in half into a training and a testing set. We optimized the tunable parameters of Chapter 8, such as the expected branching factor and projected angles, by measuring the distribution of each feature on the ground truth trees of the training set. We approximated the force field at each location using either a radially symmetric vector field centered at the root (WIDE) or a vector field in which every vector was parallel to the z-axis (RICE, SKETCH). Both vectors fields had constant magnitude. Table 12.2 summarizes all the tunable parameters of our growth model that we had to optimize for each dataset, as well as the method that we used to estimate each value.

Table 12.2: Tunable model parameters

Parameter	Description	Tuning method
p_s	Branching distribution	Fitting a one-inflated Poisson distribution to the branching statistics of the training data
$\boldsymbol{\mu}_i$	Expected growth direction of the i-th child	Mean of corresponding training data
κ_P	Von-Mises concentration parameter	Variance of corresponding training data (Khatri and Mardia, 1977)
$\mathbf{f}$	Force field	Radially symmetric or unidirectional vector field (details in text)

We then applied our greedy algorithm and our heuristic search algorithms to each planar graph. The number of possible solutions for a planar graph is an exponential function of how many faces F it has (Buchin and Schulz, 2010). Thus, for each graph we adjusted our search algorithm to explore $100F$ trees.[1] As we ran our search algorithm, we also recorded the best tree found after exploring $10F$ and $50F$ trees.

Finally, we quantified the difference between the greedy tree and the three directed trees found via our heuristic search to the ground truth using the six similarity scores defined in Chapter 11.

12.3 Results

The results for each dataset are summarized in Tables 12.3-12.11 and three example results for each dataset are shown in Figures 12.6 to 12.14. In each table, n is the number of planar graphs in the test set and $\mu(F)$ and $\sigma(F)$ are the mean and

[1] Initial experiments indicated that searching beyond $100F$ had minimal impact on the results.

Table 12.3: WIDE - Local similarities

($n = 8$, $\mu(F) = 92.7$, $\sigma(F) = 25.5$)

Method	s_p	s_f
Second human rater	0.9881 ($\pm$ 0.01)	0.9911 ($\pm$ 0.01)
Heuristic search ($100F$)	0.9662 ($\pm$ 0.02)	0.9719 ($\pm$ 0.02)
Heuristic search ($50F$)	0.9518 ($\pm$ 0.03)	0.9597 ($\pm$ 0.03)
Heuristic search ($10F$)	0.9167 ($\pm$ 0.03)	0.9292 ($\pm$ 0.04)
Greedy only	0.9059 ($\pm$ 0.03)	0.9202 ($\pm$ 0.03)

Table 12.4: WIDE - Absolute path similarities

($n = 8$, $\mu(F) = 92.7$, $\sigma(F) = 25.5$)

Method	s_a	s_l
Second human rater	0.8264 ($\pm$ 0.13)	0.7944 ($\pm$ 0.14)
Heuristic search ($100F$)	0.7316 ($\pm$ 0.15)	0.6746 ($\pm$ 0.17)
Heuristic search ($50F$)	0.6996 ($\pm$ 0.14)	0.6402 ($\pm$ 0.16)
Heuristic search ($10F$)	0.5246 ($\pm$ 0.2)	0.4475 ($\pm$ 0.17)
Greedy only	0.5071 ($\pm$ 0.19)	0.4218 ($\pm$ 0.17)

standard deviation of the number of faces per graph, respectively.

For readibility, we split the six similarities into three tables per dataset. The first table lists the local similarities (s_p, s_f); the second table includes the absolute ancestor similarities (s_a, s_l), while the last table per dataset gives the values for the relative path similarities (s_r, s_b). In each table, we have listed the mean and standard deviation for the best tree found by our heuristic search after $10F$, $50F$, and $100F$ iterations, as well as for the greedy solution. The tables for the SKETCH dataset are further subdivided into four sections: the first lists the performance of the algorithms on all the graphs in the test set, while the other three subsections show the results on each of the three levels of difficulty.

Overall, our proposed methodology accurately estimates a highly likely tree for the different input graphs in the three datasets. The local similarity scores show that

Table 12.5: WIDE - Relative path similarities

($n = 8$, $\mu(F) = 92.7$, $\sigma(F) = 25.5$)

Method	s_r	s_b
Second human rater	0.9398 ($\pm$ 0.03)	0.9284 ($\pm$ 0.04)
Heuristic search ($100F$)	0.8599 ($\pm$ 0.09)	0.8428 ($\pm$ 0.09)
Heuristic search ($50F$)	0.8521 ($\pm$ 0.07)	0.8242 ($\pm$ 0.09)
Heuristic search ($10F$)	0.7730 ($\pm$ 0.05)	0.7161 ($\pm$ 0.06)
Greedy only	0.7481 ($\pm$ 0.06)	0.6863 ($\pm$ 0.07)

Table 12.6: RICE - Local similarities

($n = 9$, $\mu(F) = 30.0$, $\sigma(F) = 9.3$)

Method	s_p	s_f
Heuristic search ($100F$)	0.9831 ($\pm$ 0.02)	0.9918 ($\pm$ 0.01)
Heuristic search ($50F$)	0.9817 ($\pm$ 0.02)	0.9905 ($\pm$ 0.01)
Heuristic search ($10F$)	0.9649 ($\pm$ 0.03)	0.9763 ($\pm$ 0.03)
Greedy only	0.9408 ($\pm$ 0.04)	0.9583 ($\pm$ 0.03)

the estimated graph generally has over 95% of the same parent-child relationships and edge orientations as the correct graph. The path similarities, on the other hand, show that the large-scale properties of the estimated tree are also very similar to the ground truth. Here, between 70-90% of the paths in the estimated graph are identical to the ones in the correct graph and, on average, between 85-98% of the edges in the paths from the root are shared by the estimated and correct trees.

Furthermore, it is important to highlight the versatility and robustness of our proposed approach. The three different datasets represent three very different types of trees that vary in their branching behavior, the expected angles between their offspring and how strongly they are influenced by surrounding forces. Also, the arborescences in the three datasets vary significantly in how close they are to being planar. Intuitively, an arborescence is close to being planar if it is very narrow along one of its axis; conversely a radially symmetric tree is very far from being planar. In

Table 12.7: RICE - Absolute path similarities

($n = 9$, $\mu(F) = 30.0$, $\sigma(F) = 9.3$)

Method	s_a	s_l
Heuristic search ($100F$)	0.8984 ($\pm$ 0.18)	0.8578 ($\pm$ 0.2)
Heuristic search ($50F$)	0.8971 ($\pm$ 0.18)	0.8567 ($\pm$ 0.2)
Heuristic search ($10F$)	0.8625 ($\pm$ 0.18)	0.8293 ($\pm$ 0.21)
Greedy only	0.7952 ($\pm$ 0.17)	0.7490 ($\pm$ 0.21)

Table 12.8: RICE - Relative path similarities

($n = 9$, $\mu(F) = 30.0$, $\sigma(F) = 9.3$)

Method	s_r	s_b
Heuristic search ($100F$)	0.9724 ($\pm$ 0.05)	0.9461 ($\pm$ 0.09)
Heuristic search ($50F$)	0.9717 ($\pm$ 0.05)	0.9455 ($\pm$ 0.09)
Heuristic search ($10F$)	0.9406 ($\pm$ 0.07)	0.9232 ($\pm$ 0.11)
Greedy only	0.8945 ($\pm$ 0.07)	0.8603 ($\pm$ 0.11)

our case, the retinal vessels were the most planar, since the retina consists of a series of flat layers. The synthetic trees, on the other hand, were the least planar since the parameters of the tree modeling software favored radially symmetric trees. In spite of these structural differences between datasets, after properly tuning our model's parameters, our methods were generally able to accurately approximate the correct solution. We now discuss each dataset in more detail.

12.3.1 WIDE dataset

In this dataset, our algorithm was able to closely approximate the performance of an expert human rater on a set of challenging retinal images with minimal specialized domain knowledge. That is, aside from the branching factor and expected angle statistics that we obtained from the training set, we did not include any other image features, such as vessel dilation or color. We expect that incorporating these additional features should further improve our results on traditional fundus images.

Table 12.9: SKETCH - Local similarities

($n = 45$, $\mu(F) = 37.68$, $\sigma(F) = 24.37$)

Subset	Method	s_p	s_f
All	Heuristic search ($100F$)	0.9551 ($\pm$ 0.03)	0.9927 ($\pm$ 0.01)
	Heuristic search ($50F$)	0.9546 ($\pm$ 0.03)	0.9920 ($\pm$ 0.02)
	Heuristic search ($10F$)	0.9516 ($\pm$ 0.04)	0.9901 ($\pm$ 0.02)
	Greedy only	0.9354 ($\pm$ 0.04)	0.9823 ($\pm$ 0.02)
Complex	Heuristic search ($100F$)	0.9483 ($\pm$ 0.03)	0.9868 ($\pm$ 0.02)
	Heuristic search ($50F$)	0.9470 ($\pm$ 0.04)	0.9850 ($\pm$ 0.03)
	Heuristic search ($10F$)	0.9404 ($\pm$ 0.04)	0.9805 ($\pm$ 0.03)
	Greedy only	0.9232 ($\pm$ 0.03)	0.9718 ($\pm$ 0.03)
Medium	Heuristic search ($100F$)	0.9630 ($\pm$ 0.02)	0.9971 ($\pm$ 0.01)
	Heuristic search ($50F$)	0.9628 ($\pm$ 0.02)	0.9970 ($\pm$ 0.01)
	Heuristic search ($10F$)	0.9604 ($\pm$ 0.02)	0.9956 ($\pm$ 0.01)
	Greedy only	0.9399 ($\pm$ 0.03)	0.9842 ($\pm$ 0.02)
Simple	Heuristic search ($100F$)	0.9540 ($\pm$ 0.04)	0.9941 ($\pm$ 0.01)
	Heuristic search ($50F$)	0.9540 ($\pm$ 0.04)	0.9941 ($\pm$ 0.01)
	Heuristic search ($10F$)	0.9540 ($\pm$ 0.04)	0.9941 ($\pm$ 0.01)
	Greedy only	0.9432 ($\pm$ 0.04)	0.9911 ($\pm$ 0.01)

However, some novel imaging systems, such as the optical coherence tomography method described in (Hendargo et al., 2013), are able to capture retinal vasculature images at a higher resolution that conventional fundus cameras. The vessels in these higher resolution images often have no color and the smaller vessels in them have uniform dilation due to the imaging system's diffraction limit. Thus, the features described above are no longer informative. However, since our method does not rely on conventional vessel features, we expect that we should also obtain good results with images from these next-generation imaging systems.

Furthermore, although the WIDE dataset had the highest mean number of faces, our algorithm obtained better parent similarity and absolute ancestor similarity re-

Table 12.10: SKETCH - Absolute path similarities

($n = 45$, $\mu(F) = 37.68$, $\sigma(F) = 24.37$)

Subset	**Method**	s_a	s_l
All	Heuristic search ($100F$)	0.6837 ($\pm$ 0.24)	0.6215 ($\pm$ 0.26)
	Heuristic search ($50F$)	0.6857 ($\pm$ 0.24)	0.6230 ($\pm$ 0.26)
	Heuristic search ($10F$)	0.6793 ($\pm$ 0.24)	0.6177 ($\pm$ 0.26)
	Greedy only	0.6470 ($\pm$ 0.24)	0.5785 ($\pm$ 0.26)
Complex	Heuristic search ($100F$)	0.6364 ($\pm$ 0.22)	0.5884 ($\pm$ 0.26)
	Heuristic search ($50F$)	0.6363 ($\pm$ 0.22)	0.5884 ($\pm$ 0.26)
	Heuristic search ($10F$)	0.6284 ($\pm$ 0.21)	0.5669 ($\pm$ 0.28)
	Greedy only	0.6026 ($\pm$ 0.22)	0.5474 ($\pm$ 0.24)
Medium	Heuristic search ($100F$)	0.6367 ($\pm$ 0.28)	0.5769 ($\pm$ 0.28)
	Heuristic search ($50F$)	0.6367 ($\pm$ 0.28)	0.5769 ($\pm$ 0.29)
	Heuristic search ($10F$)	0.6253 ($\pm$ 0.28)	0.5669 ($\pm$ 0.28)
	Greedy only	0.5941 ($\pm$ 0.28)	0.5247 ($\pm$ 0.29)
Simple	Heuristic search ($100F$)	0.7842 ($\pm$ 0.21)	0.7036 ($\pm$ 0.25)
	Heuristic search ($50F$)	0.7842 ($\pm$ 0.21)	0.7036 ($\pm$ 0.25)
	Heuristic search ($10F$)	0.7842 ($\pm$ 0.21)	0.7036 ($\pm$ 0.25)
	Greedy only	0.7443 ($\pm$ 0.22)	0.6635 ($\pm$ 0.25)

sults for this dataset than for the SKETCH dataset. We speculate that this is because retinal vessels are very close to being planar. Thus, there is often very little difference between the original and the projected angles, which makes the prior on the angles between siblings very informative.

12.3.2 RICE dataset

We obtained the best results on this dataset relative to the other two. The difference is particularly striking for the absolute path similarities. On average, around 90% of the paths in the estimated tree were identical to the correct tree, compared to around 70-75% for the best results in the other datasets. These values indicate that

Table 12.11: SKETCH - Relative path similarities

($n = 45$, $\mu(F) = 37.68$, $\sigma(F) = 24.37$)

Subset	**Method**	s_r	s_b
All	Heuristic search ($100F$)	0.9167 ($\pm$ 0.07)	0.9000 ($\pm$ 0.08)
	Heuristic search ($50F$)	0.9170 ($\pm$ 0.07)	0.8997 ($\pm$ 0.08)
	Heuristic search ($10F$)	0.9127 ($\pm$ 0.08)	0.8957 ($\pm$ 0.09)
	Greedy only	0.8736 ($\pm$ 0.09)	0.8591 ($\pm$ 0.1)
Complex	Heuristic search ($100F$)	0.9205 ($\pm$ 0.07)	0.9171 ($\pm$ 0.07)
	Heuristic search ($50F$)	0.9205 ($\pm$ 0.07)	0.9168 ($\pm$ 0.07)
	Heuristic search ($10F$)	0.9108 ($\pm$ 0.08)	0.9076 ($\pm$ 0.08)
	Greedy only	0.8555 ($\pm$ 0.08)	0.8575 ($\pm$ 0.08)
Medium	Heuristic search ($100F$)	0.9117 ($\pm$ 0.07)	0.9022 ($\pm$ 0.08)
	Heuristic search ($50F$)	0.9126 ($\pm$ 0.07)	0.9015 ($\pm$ 0.08)
	Heuristic search ($10F$)	0.9095 ($\pm$ 0.07)	0.8988 ($\pm$ 0.08)
	Greedy only	0.8681 ($\pm$ 0.1)	0.8586 ($\pm$ 0.09)
Simple	Heuristic search ($100F$)	0.9178 ($\pm$ 0.08)	0.8807 ($\pm$ 0.1)
	Heuristic search ($50F$)	0.9178 ($\pm$ 0.08)	0.8807 ($\pm$ 0.1)
	Heuristic search ($10F$)	0.9178 ($\pm$ 0.08)	0.8807 ($\pm$ 0.1)
	Greedy only	0.8974 ($\pm$ 0.09)	0.8612 ($\pm$ 0.12)

most of the errors in this dataset were located at the periphery as opposed to closer to the root.

We speculate that these better results were primarily due to two factors: first, the graphs corresponding to the rice plants had fewer faces than the other datasets, particularly compared to the retinal graphs. Furthermore, although the rice roots were quite radially symmetric, they also had a strong tendency to grow downwards towards the ground, which made the projected angles between siblings less variable than in the SKETCH dataset.

12.3.3 SKETCH dataset

In this dataset, the difference between the parent and flow similarities was more pronounced than in the other two datasets. We speculate that this is because, due to the simulated limited resolution we imposed on the projections, the graphs in this dataset had more instances of vertices of degree five or higher than the other two. For these vertices, it is often easy to determine the orientation of their adjacent edges if the edges are well aligned with the expected direction of growth. However, determining which of the incoming edges is the parent of which of the outgoing edges is generally far more challenging, because there is often little difference in the projected angles between edges which adjacent in the original tree compared to edges which are not.

Furthermore, the results on the three difficulty subsets give us further insight into the relationship between the local and global properties of a tree. Interestingly, the local and relative similarities were very similar for the three subsets. However, although the local similarities were actually best for the graphs in the medium difficulty subset, this subset had the worst relative path similarities out of the three subsets. This suggests that the algorithm had fewer errors in the periphery and more errors closer to the root for the medium subset relative to the other two. This difference in where the errors were located is particularly clear in terms of the absolute path similarities. Here, the algorithm made far fewer errors near the root in the simple subset compared to the other two, although it made about the same number of errors overall in the three subsets.

12.4 Discussion

As noted above, our algorithms accurately estimate the most likely tree under a wide variety of settings. To shed further light on our proposed approach, we now analyze

some key aspects of our overall results in greater detail.

12.4.1 Estimation complexity

As noted above, the number of possible solutions is an exponential function of the number of faces. Thus, all else being equal, the estimation problem should become significantly more difficult as F grows larger. However, in general our algorithms were able to obtain similar results for graphs with vastly different numbers of solutions. For instance, the retinal graphs have around three times as many faces as the rice graphs and between two to four times as many as the the differetn subsets of the SKETCH dataset. Thus, the spaces of possible solutions for the graphs in the WIDE dataset are vastly larger than for the other two. Nevertheless, the parent similarities between our estimated trees and the ground truth were almost as good for the WIDE graphs as for the RICE graphs and even better than for the SKETCH graphs.

Furthermore, aside from the absolute path similarities, the results for each of the subsets of the SKETCH dataset were very similar. This suggests that the estimation problem did not grow significantly harder as the complexity of the input graphs increased.

The above analysis suggests that although the tree estimation problem is NP-hard in the worst case, our tree growth model strongly constraints the set of realistic solutions and allows us to directly hone in on the most likely trees.

12.4.2 Parameter robustness

Our algorithms determine the prior probability of every potential solution using the tree growth model presented in Chapter 7 and 8. That is, for every vertex $\mathbf{v} \in T$, its local probability is determined by three factors:

1. Its branching factor

2. The angles between its children

3. The direction of growth of its children relative to the force field.

To better understand the relative importance of these three factors, we ran a series of additional experiments in which we only considered one or two of the these factors in turn, as described below.

We first selected two WIDE images whose parent similarities (0.9692,0.9744) were close to the mean of the test set (0.9662). We then ran our tree estimation algorithms under different conditions. In each condition, we ignored one or two of the three factors outlined above. Our six conditions were:

1. Branching factor only (p_s)
2. Sibling angles only ($\mathbf{s}(\mathbf{v})$)
3. Force field only ($\mathbf{f}(\mathbf{v})$)
4. Branching and sibling angles only (No $\mathbf{f}(\mathbf{v})$)
5. Branching and force field only (No $\mathbf{s}(\mathbf{v})$)
6. Sibling angles and force field (No p_s).

We ran our search algorithm under each condition until it explored $200F$ trees. Tables 12.12 to 12.13 list the mean results for the two trees for each condition at different search depths in terms of the parent, absolute ancestor and relative ancestor similarities.

Overall, it seems that the angles between siblings is the most informative of the three features, while the force field is the least. The algorithm was able to obtain consistently better results when it incorporated the former feature than when it did not. In particular, the absolute ancestor scores are far worse when angles between siblings are ignored. Furthermore, omitting the branching factor had a

Table 12.12: WIDE - Parameter importance - s_p

Method	p_s	$\mathbf{s}(\mathbf{v})$	$\mathbf{f}(\mathbf{v})$	No p_s	No $\mathbf{s}(\mathbf{v})$	No $\mathbf{f}(\mathbf{v})$
Search ($200F$)	0.9468	0.9638	0.9208	0.9646	0.9352	0.9648
Search ($150F$)	0.9468	0.9621	0.9208	0.9646	0.9272	0.9588
Search ($100F$)	0.9427	0.9704	0.9236	0.9630	0.9296	0.9585
Search ($50F$)	0.9401	0.9551	0.9296	0.9557	0.9333	0.9515
Search ($10F$)	0.9213	0.9215	0.9060	0.9207	0.9180	0.9265
Greedy only	0.9038	0.9108	0.8994	0.9100	0.9030	0.9109

Table 12.13: WIDE - Parameter importance - s_a

Method	p_s	$\mathbf{s}(\mathbf{v})$	$\mathbf{f}(\mathbf{v})$	No p_s	No $\mathbf{s}(\mathbf{v})$	No $\mathbf{f}(\mathbf{v})$
Search ($200F$)	0.3699	0.6259	0.2472	0.6511	0.2503	0.7482
Search ($150F$)	0.3982	0.6259	0.2472	0.6511	0.2523	0.7377
Search ($100F$)	0.3896	0.6488	0.2512	0.6511	0.2523	0.7358
Search ($50F$)	0.3875	0.6310	0.2512	0.6666	0.2543	0.7268
Search ($10F$)	0.3199	0.6066	0.2954	0.6066	0.3377	0.7071
Greedy only	0.3145	0.5985	0.2959	0.5985	0.3458	0.6961

larger negative effect, particularly in terms of the absolute ancestor similarity, than omitting the force field, which suggests the former feature is more informative.

We speculate that the force field is the weakest feature because we were only able to roughly approximate it. As noted above, we determined the preferred direction of growth at a given location using simple, constant vector fields, which were not able to capture location-dependent differences in the preferred direction of growth. For instance, although most retinal vessels tend to grow directly away from the optic nerve, the small vessels near the macula instead grow towards this second retinal landmark. We expect that using more accurate, domain-specific force fields will make this feature more informative.

Table 12.14: WIDE - Parameter importance - s_r

Method	p_s	$\mathbf{s}(\mathbf{v})$	$\mathbf{f}(\mathbf{v})$	No p_s	No $\mathbf{s}(\mathbf{v})$	No $\mathbf{f}(\mathbf{v})$
Search ($200F$)	0.7653	0.8180	0.5699	0.7861	0.5910	0.8710
Search ($150F$)	0.7925	0.8105	0.5699	0.7861	0.5558	0.8670
Search ($100F$)	0.7896	0.8887	0.5723	0.7786	0.5587	0.8529
Search ($50F$)	0.7891	0.8559	0.5864	0.8320	0.5840	0.8453
Search ($10F$)	0.7926	0.8414	0.5830	0.8328	0.6204	0.8441
Greedy only	0.7590	0.8300	0.6109	0.8213	0.6385	0.8312

12.4.3 Feature ambiguity

Estimating a tree's topology involves weighing a combination of features. Generally, these features are well-correlated; for instance the most likely angles between siblings tend to occur between edges that are well aligned with the surrounding force field. In practice, however, this correlation does not always hold. When the features are ambiguous, the tree estimation problem becomes markedly more difficult.

Figure 12.15 illustrates such a dilemma faced by the algorithm. Here, the direction of growth of the central vessel is towards the optic nerve and not away from it. In this instance, the algorithm had to choose between two low-probability options: (1) flowing towards the root or (2) assuming the branch-point further away from the root was a branch-crossing. We expect that incorporating more domain-dependent features, such as vessel dilation, will allow our algorithm to better disambiguate these difficult cases.

In the following chapter, we will explore further potential improvements and extensions to our current tree estimation methodology that may better allow us to estimate the most likely tree.

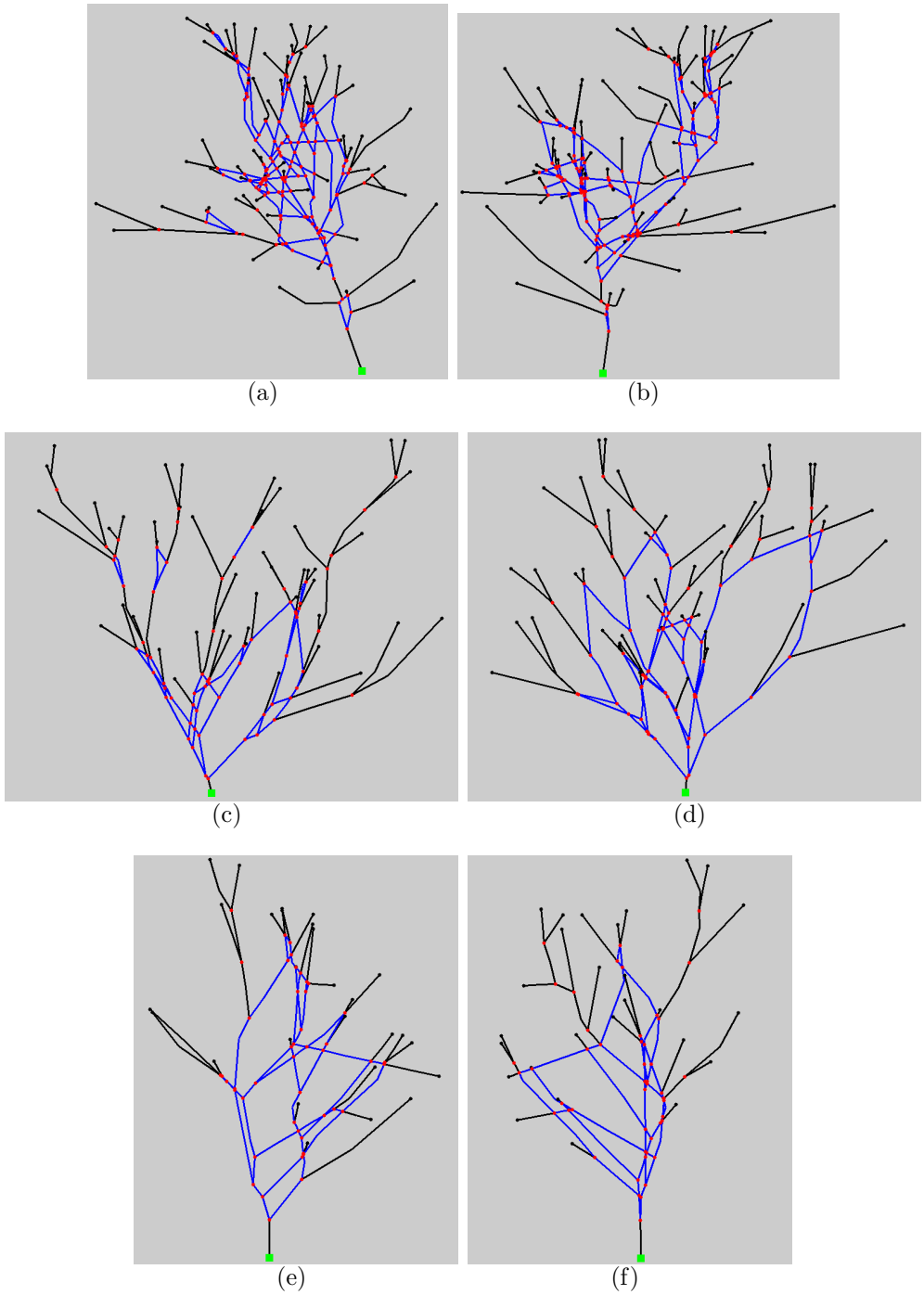
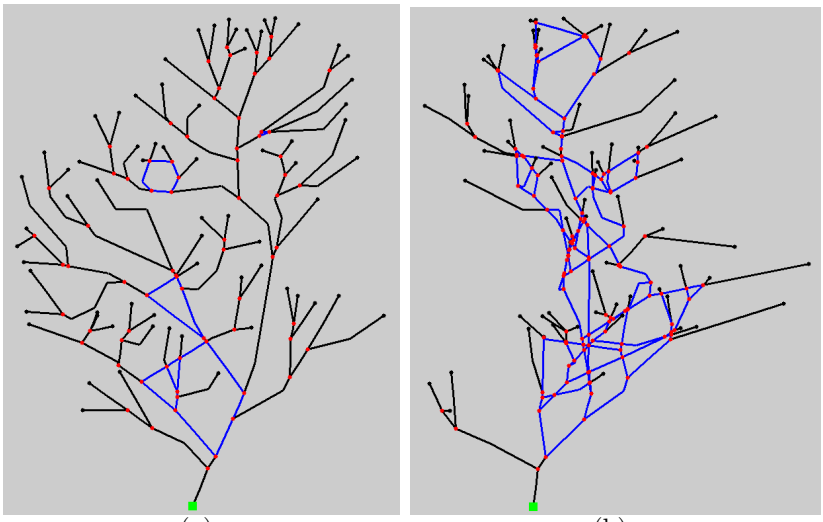


FIGURE 12.4: **SKETCH dataset:** (This Figure is best viewed on-screen.) Each row consists of two projections from the same arborescence. The root is marked in green, the branch-points in red and the end-points in black; the edges which are part of a circuit are highlighted in blue. **(a,b)** are from the complex subset, **(c,d)** are from the medium, and **(e,f)** are from the simple subset. All images are shown at the same scale.



(a) (b)

FIGURE 12.5: **SKETCH projection complexity:** (This Figure is best viewed on-screen.) Two projections from the same graph can result in planar graphs that have markedly different complexities. **(a)** has 7 faces, while **(b)** has 71 faces. The edges which are part of a circuit are highlighted in blue.

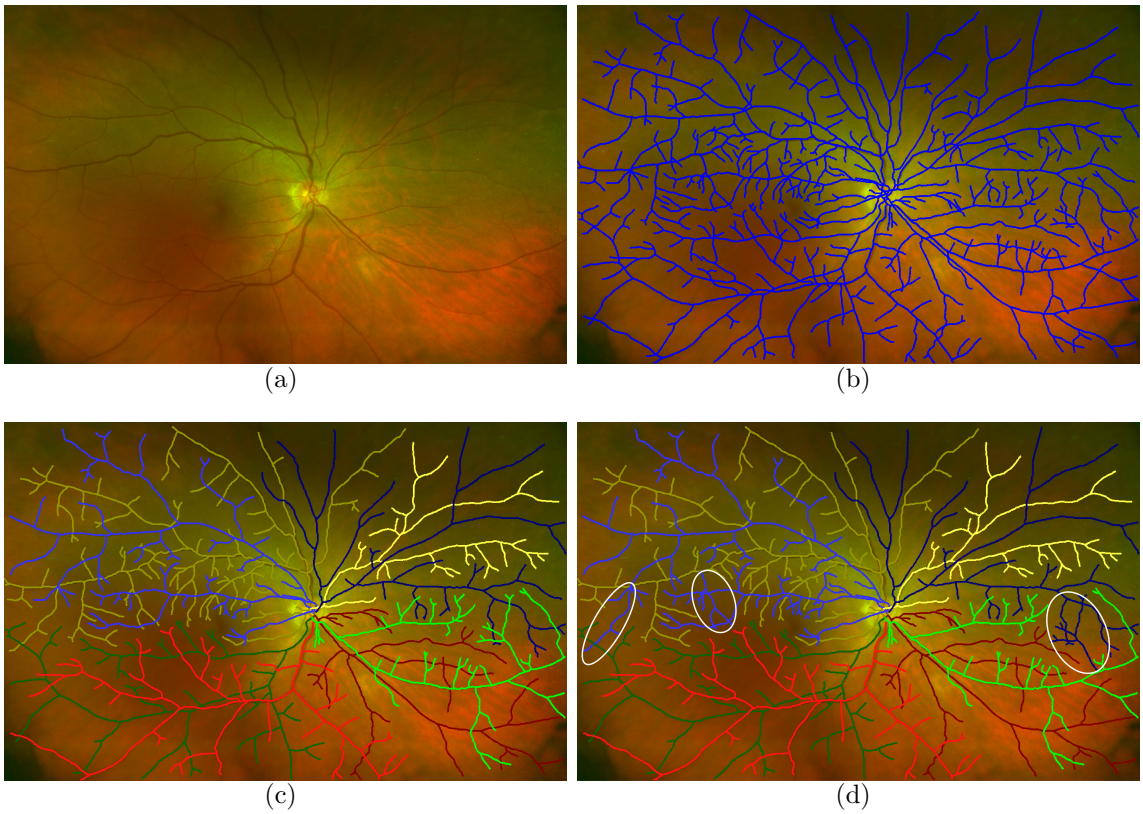


FIGURE 12.6: **Retinal vessel example 1:** (This Figure is best viewed on-screen.) **(a)** An input retinal image. **(b)** The extracted graph. **(c)** The ground truth tree **(d)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The similarity scores are: $s_f = 0.978$, $s_p = 0.974$, $s_r = 0.955$, $s_b = 0.944$, $s_a = 0.964$, $s_l = 0.951$.

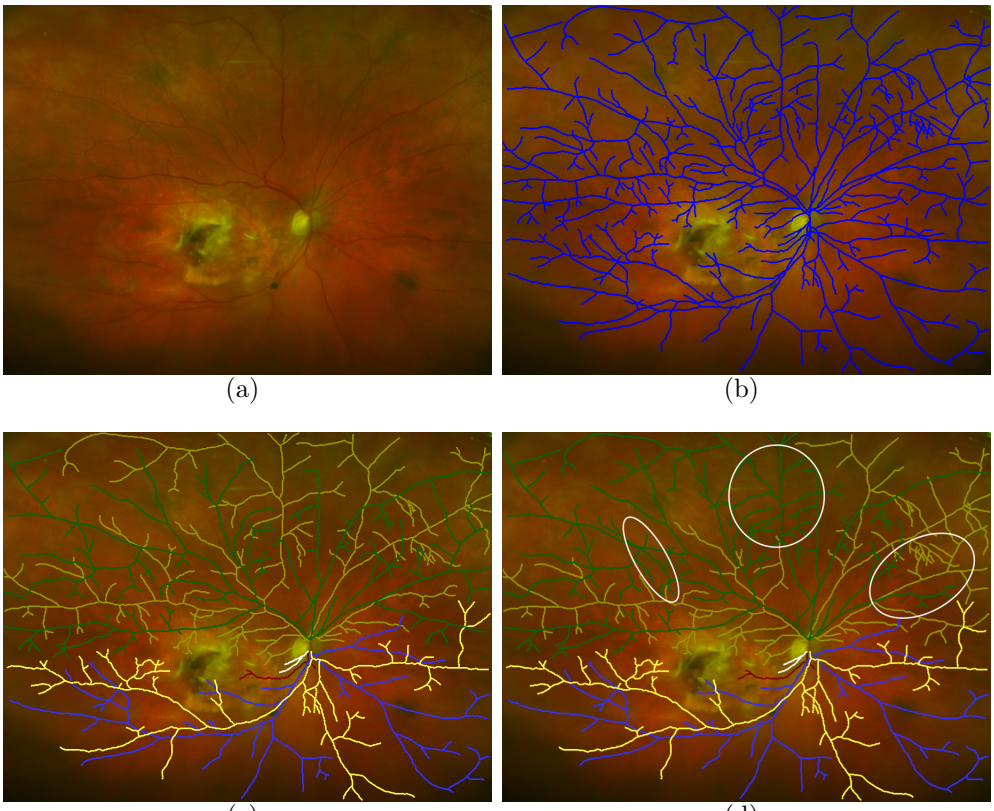


FIGURE 12.7: **Retinal vessel example 2:** (This Figure is best viewed on-screen.) **(a)** An input retinal image. **(b)** The extracted graph. **(c)** The ground truth tree **(d)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The similarity scores are: $s_f = 0.979$, $s_p = 0.969$, $s_r = 0.847$, $s_b = 0.830$, $s_a = 0.800$, $s_l = 0.516$.

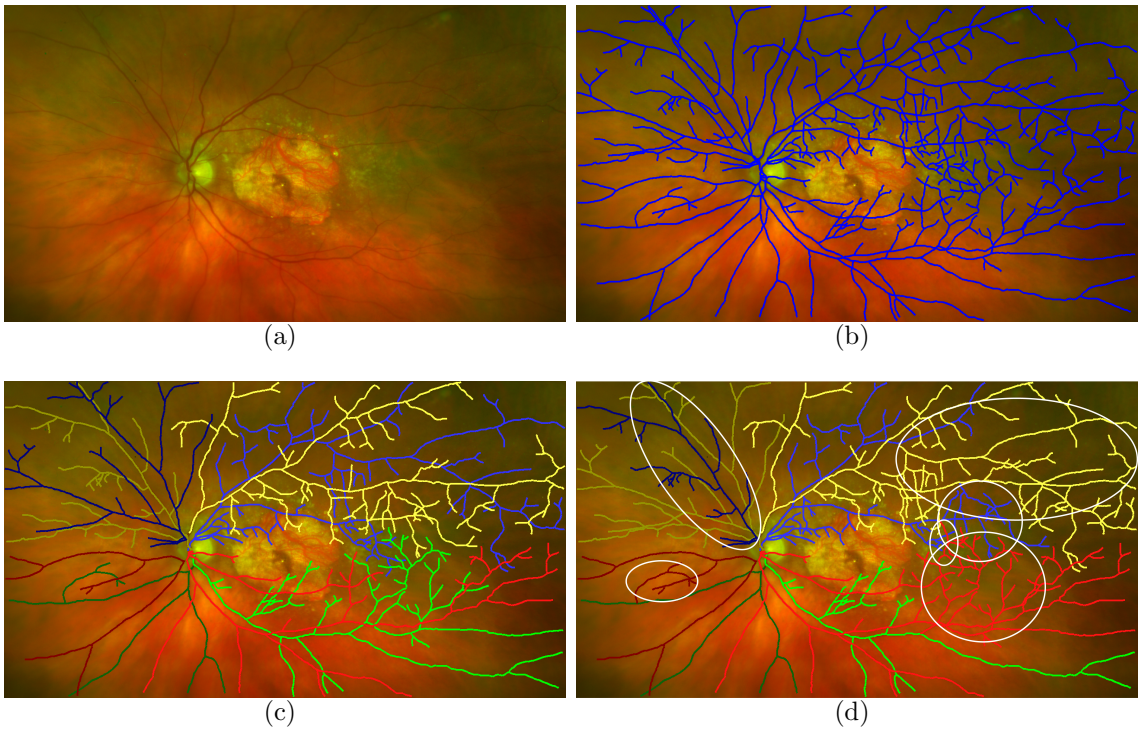


FIGURE 12.8: **Retinal vessel example 3:** (This Figure is best viewed on-screen.) **(a)** An input retinal image. **(b)** The extracted graph. **(c)** The ground truth tree **(d)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The similarity scores are: $s_f = 0.924$, $s_p = 0.914$, $s_r = 0.673$, $s_b = 0.627$, $s_a = 0.630$, $s_l = 0.619$.

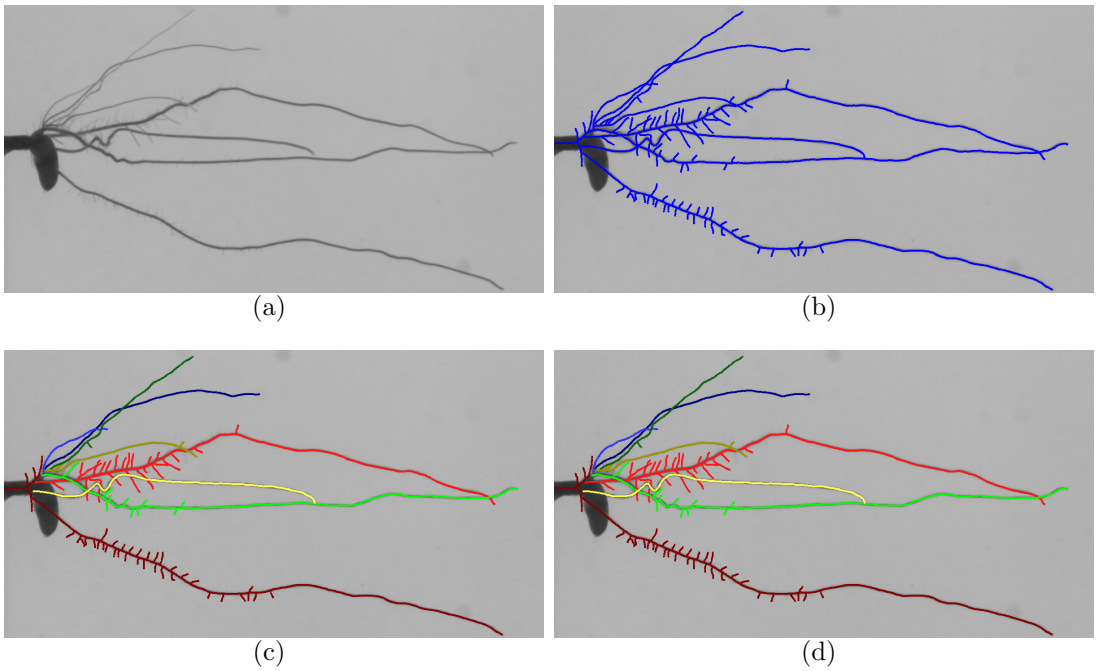


FIGURE 12.9: **Rice plant example 1:** (This Figure is best viewed on-screen.) **(a)** An input plant root image. **(b)** The extracted graph. **(c)** The ground truth tree **(d)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. Here, our method estimated the correct tree. The images have been rotated 90° for easier visualization. The similarity scores are: $s_f = 1$, $s_p = 1$, $s_r = 1$, $s_b = 1$, $s_a = 1$, $s_l = 1$.

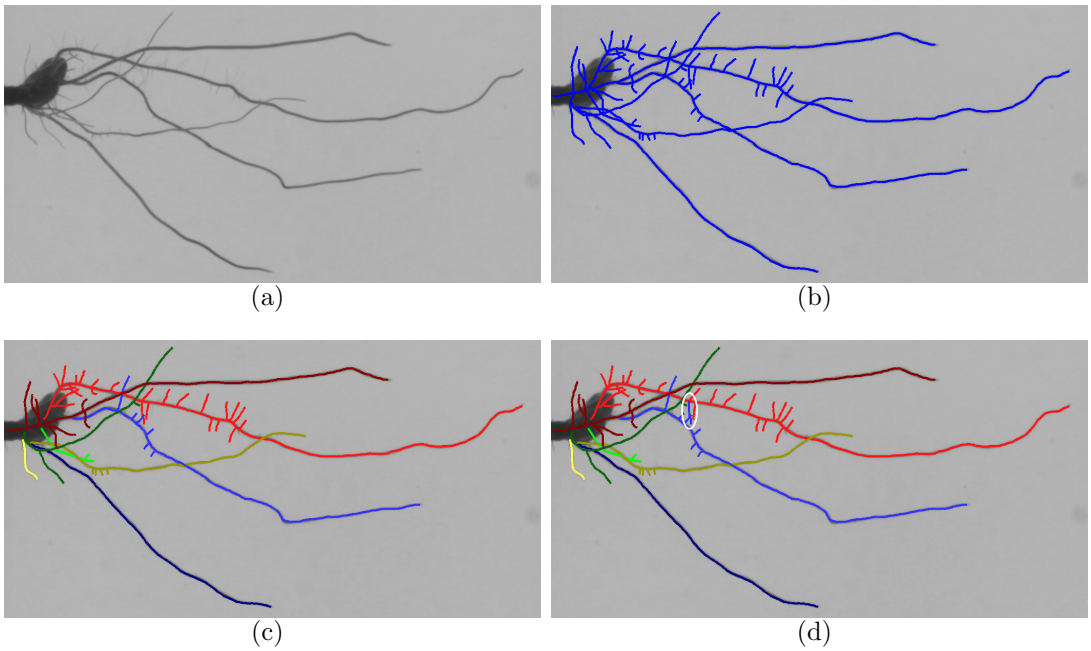
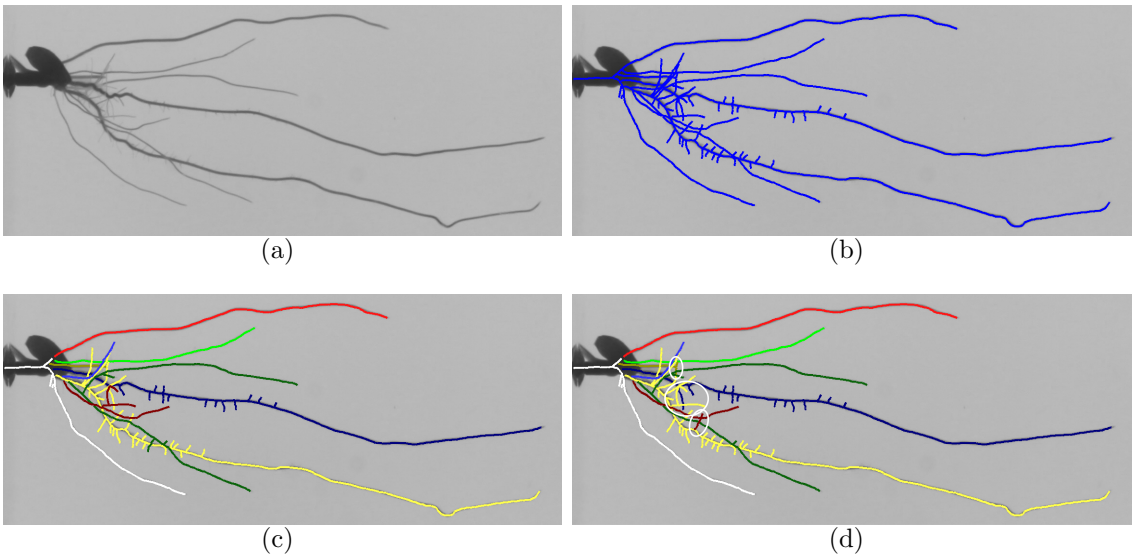
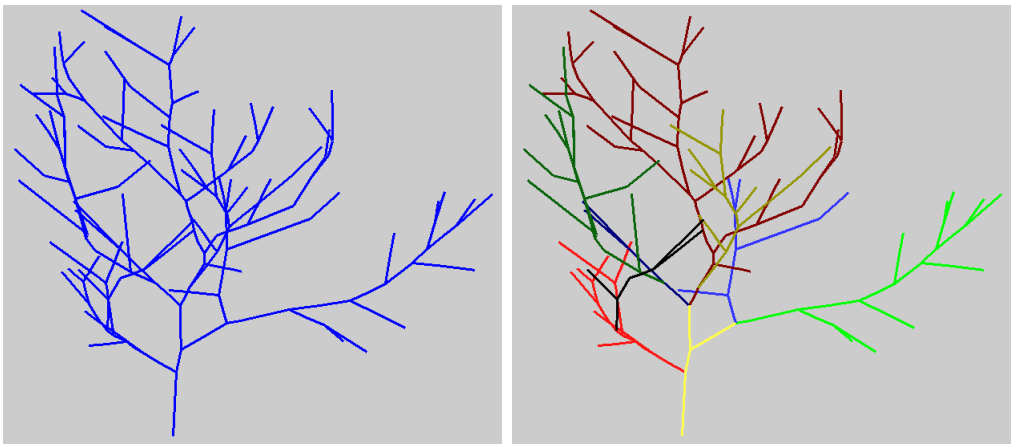


FIGURE 12.10: **Rice plant example 2:** (This Figure is best viewed on-screen.) **(a)** An input plant root image. **(b)** The extracted graph. **(c)** The ground truth tree **(d)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The sole error in the estimated tree is highlighted with a white ellipse. The images have been rotated 90° for easier visualization. The similarity scores are: $s_f = 0.998$, $s_p = 0.995$, $s_r = 0.996$, $s_b = 0.992$, $s_a = 0.985$, $s_l = 0.961$.



(a) (b) (c) (d)

FIGURE 12.11: **Rice plant example 3:** (This Figure is best viewed on-screen.) **(a)** An input plant root image. **(b)** The extracted graph. **(c)** The ground truth tree **(d)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The images have been rotated 90° for easier visualization. The similarity scores are: $s_f = 0.969$, $s_p = 0.940$, $s_r = 0.891$, $s_b = 0.876$, $s_a = 0.879$, $s_l = 0.742$.



(a) (b)

(c)

FIGURE 12.12: **Synthetic tree example 1:** (This Figure is best viewed on-screen.) **(a)** The projected graph. **(b)** The ground truth tree **(c)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The similarity scores are: $s_f = 1$, $s_p = 0.987$, $s_a = 0.9177$, $s_l = 0.932$, $s_r = 1$, $s_b = 1$.

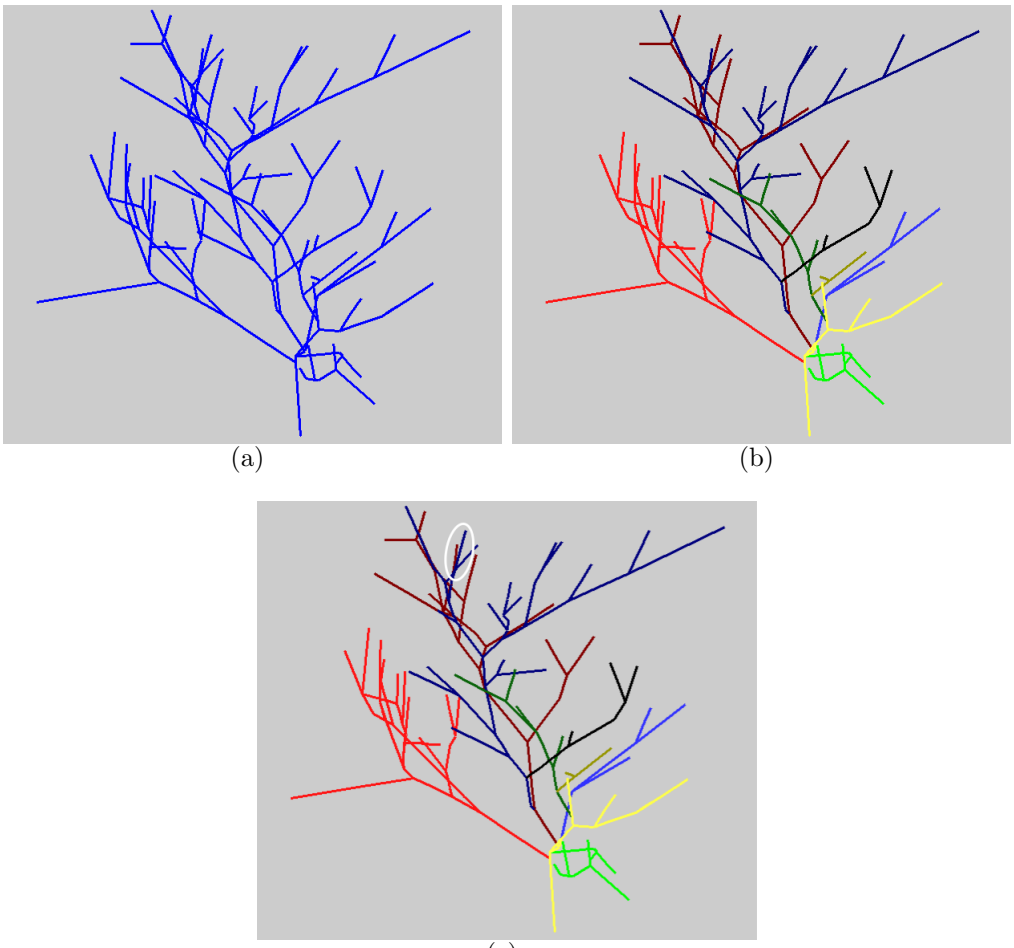


FIGURE 12.13: **Synthetic tree example 2:** (This Figure is best viewed on-screen.) **(a)** The projected graph. **(b)** The ground truth tree **(c)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The similarity scores are: $s_f = 0.981$, $s_p = 0.928$, $s_a = 0.837$, $s_l = 0.750$, $s_r = 0.959$, $s_b = 0.919$.

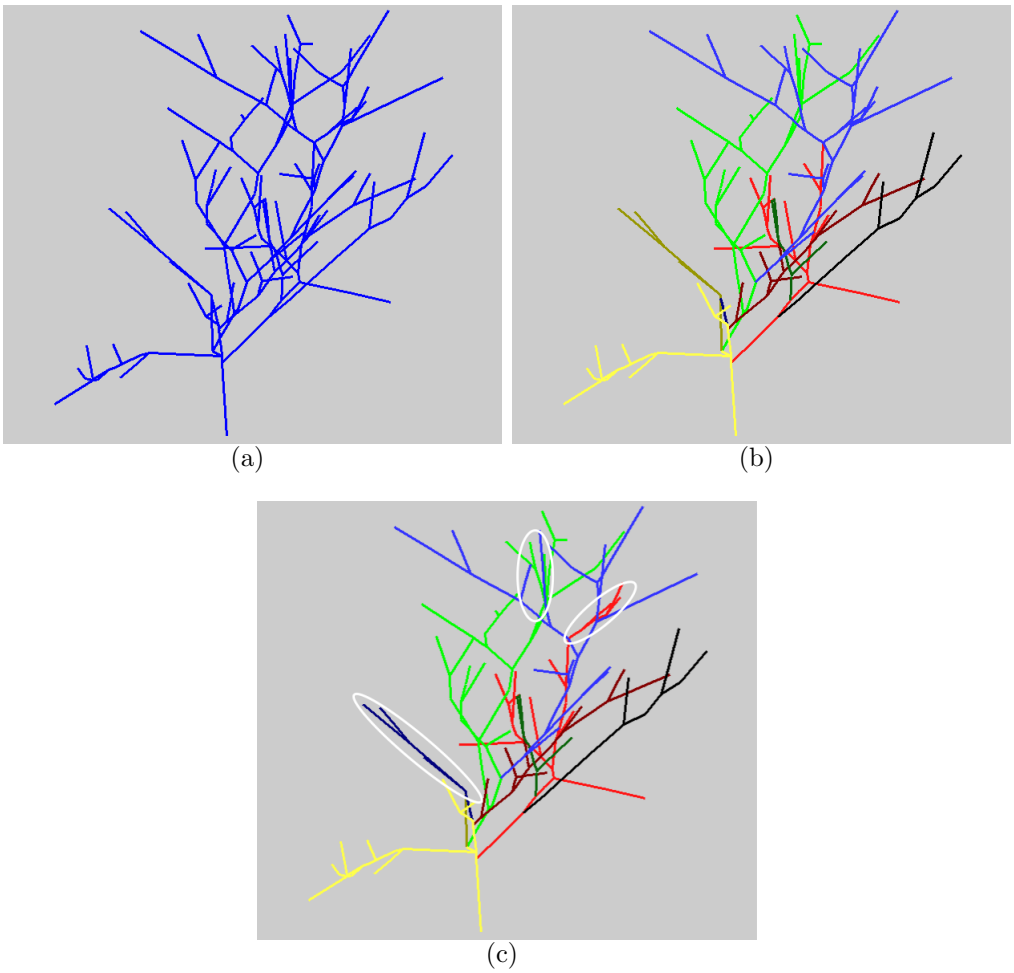
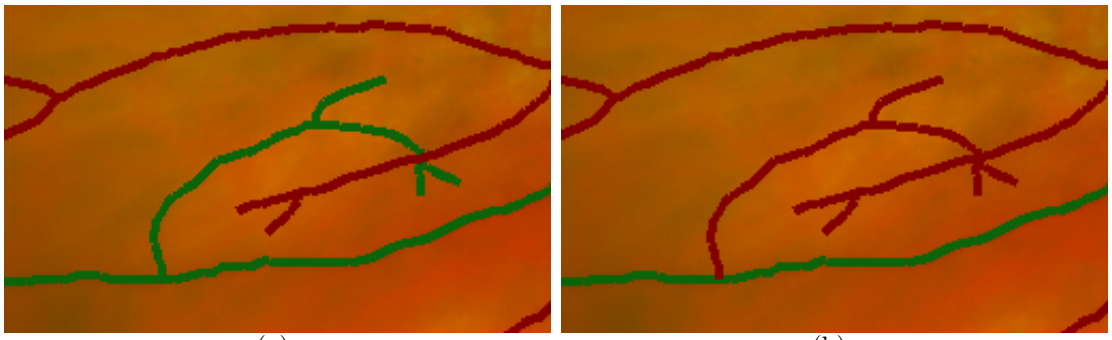


FIGURE 12.14: **Synthetic tree example 3:** (This Figure is best viewed on-screen.) **(a)** The projected graph. **(b)** The ground truth tree **(c)** The best tree found by our heuristic search. Each of the different colors indicates a different sub-tree of the estimated tree. The white ellipses highlight when two sub-tree differ in the two trees. The similarity scores are: $s_f = 0.999$, $s_p = 0.967$, $s_a = 0.585$, $s_l = 0.518$, $s_r = 0.948$, $s_b = 0.901$.



(a) (b)

FIGURE 12.15: **Feature ambiguity:** (This Figure is best viewed on-screen.) Estimating a tree's topology involves weighing a combination of features. In this instance, the algorithm had to choose between two low-probability options: (1) flowing towards the root or (2) assuming the branch-point further away from the root was a branch-crossing. **(a)** The ground truth tree. **(b)** The best tree found by our heuristic search.

13

Future Work

In the previous chapters we detailed a methodology for estimating the most likely topology of a three-dimensional tree given a single two-dimensional image. The experiments in Chapter 12 show that our tree estimation algorithms accurately estimate the most likely topology of various types of trees, including retinal vessels and plant roots. In this chapter we explore various potential improvements to our methodology that may allow us to better estimate the most likely tree.

More concretely, this chapter explores three extensions to our work: the first two sections explore how to make our algorithms more robust to errors in the input graph. In the first section, we outline an expanded search algorithm in which we apply our heuristic tree search algorithm not only to the input graph, but to other planar graphs which are similar to it. Then, in the following section we propose a novel graph extraction method that combines the method described in Appendix C with the tree growth model defined in Chapter 7. Our enhanced graph extraction approach incorporates stronger shape and topology priors which should allow it to estimate the initial planar graph more accurately.

The third section introduces a novel sampling technique with which we can obtain

different, but potentially still likely greedy solutions. Using multiple initial starting locations may allow our heuristic search to obtain a good final estimate even if the greedy solution is poor.

Finally, in the last section, we will explore branch ordering and artery/vein classification, two extensions to our methodology which are relevant to various retinal vessel applications.

13.1 Robust tree estimation

As noted in Chapter 3, the main limitation of our current approach is that state-of-the-art graph extraction techniques, such as (Türetken et al., 2010, 2011; Yedidya and Hartley, 2008; Rattathanapad et al., 2012) and our method described in Appendix C, are not robust enough to consistenly estimate the correct planar graph from noisy images.

So far, however, our tree estimation algorithms implicitly assume that the input graph G is a faithful representation of the original arborescence A, since they only modify the connectivity between the edges of G. In general, however, the planar graph is itself estimated from a noisy image and may contains errors such as missing or spurious edges.

In Chapter 3, we formulated our tree estimation problem as a special case of Maximum A Posteriori (MAP) estimation,

$$A^\star = \operatorname*{argmax}_{A \in \mathcal{A}} p_O(G|A) p_M(A)$$

where the observation probability $p_O(G|A)$ is uniform (proportional to a constant) and only arborescences whose projections are isomorphic to G have non-zero probability. In this case, we leave all regularization up to the prior model M. Here, on the other hand, we propose a way of defining the observation probability in terms of *inexact graph matching*, as defined below.

13.1.1 Inexact graph matching

In many applications, such as object recognition and sensor fusion, we wish to determine whether two or more graphs extracted from some input data represent the same object. *Graph matching* is the general problem of determining how similar two graphs are. Exact graph matching reduces to checking for graph isomorphism. Two graphs G and H are said to be *isomorphic* if there is a bijection $f : V(G) \mapsto V(H)$ between their vertices such that adjacency is preserved (Fortin, 1996). In other words,

$$(u, v) \in E(G) \Leftrightarrow (f(u), f(v)) \in E(H), \forall u, v \in G.$$

In practice, however, the graphs extracted from noisy data will contain errors. Here, two graphs that represent the same object may not be isomorphic. In this case, we seek the best inexact matching between the two graphs. Arguably the most common approach for defining the similarity between a pair of graphs is to use a *graph edit distance* (GED) (Gao et al., 2010). Here, similarity is defined in terms of how much we have to change one graph to transform it into another.

More concretely, any non-embedded, finite graph G can be converted to a different graph H by a finite sequence of the following atomic operations:

- Adding a vertex.
- Deleting a vertex.
- Adding an edge.
- Deleting an edge.

The first operation adds a vertex with no edges to the graph; the second removes a vertex and all its incident edges; the third operation adds an edge between two existing vertices and the fourth deletes an edge between two vertices in the graph.

The above operations can be applied to both directed and undirected graphs. Here we will focus on undirected graphs, but our discussion will also apply in the directed case.

It is not hard to see that more complex editing operations, such as edge contractions (Diestel, 2010) and the partitions defined Chapter 4 can be reduced to applying a sequence of the above four atomic edits. Therefore, analogously to the flipping operation defined in Chapter 5, these editing operations induce a connected *meta-digraph* over the set of all possible (undirected or directed) graphs. Here, G has an incoming edge from graph H if and only G can be obtained from H by applying exactly one of the four operations listed above.

Note that the meta-digraph is directed because the adding and deleting operations are not symmetric; deleting a vertex of degree four requires one operation, while adding an equivalent vertex requires five operations (one vertex addition and four edge additions). On the other hand, it is easy to see that the meta-digraph is connected by noting that all graphs can be converted to and from the null graph (the graph with no vertices or edges).

In terms of graph matching, the similarity between two graphs is then defined as the shortest path between two graphs in the meta-digraph, where each edit operation may be assigned a different weight using a weighing function f. Unfortunately, computing this distance, even for uniform f, is NP-hard (Zeng et al., 2009) because it can be used to solve the subgraph isomorphism problem, a classic NP-complete problem.

For our purposes, however, we can make use of the atomic editing operations defined above to expand our tree estimation search to include other planar graphs which are similar to our initial noisy graph. First, however, we need to define an equivalent meta-digraph for *embedded* graphs.

13.1.2 Embedded meta-digraph

The vertices in the meta-digraph defined above are *non-embedded* graphs. In an embedded graph, each vertex is defined not only by its connectivity, but also by its location. We can easily extend the atomic edit operations defined above to also include an embedding. In this case, when we add a new vertex we also have to specify its location. However, if a vertex's location is defined in $\mathbb{R}^2$, then the set of possible new graphs after a single vertex addition is infinite. Therefore, it is not possible to define an analogous meta-graph for embedded graphs, unless we somehow constrain the vertex addition operation.

We propose to constrain this operation by allowing new vertices to only be drawn from a finite set of *alternate* vertices V_{alt}. V_{alt} is a set of embedded vertices which are not part of the original graph, i.e. $V_{\text{alt}} \cap V(G) = \varnothing$. Here, we restrict the vertex addition operation so that it can only add vertices from V_{alt} to G.

Given a total set of allowed vertices $V_{\text{alt}} \cup V(G)$, we can now define an embedded meta-graph over the set of possible graphs that include some subset of the allowed vertices. In other words, every graph in the embedded meta-digraph is a subgraph of the complete graph over the vertex set $V_{\text{alt}} \cup V(G)$.

The above formulation is similar to the k-MST problem (Chimani et al., 2010), which seeks a minimum spanning subtree that spans k vertices. In our case, however, we do not restrict the edited graphs to be spanning trees, but require them to be planar.

13.1.3 Expanded observation model

We are now ready to expand our observation probability $p_O(G|A)$ to the case where G and $P(A)$ are not isomorphic. Let G_A be the noise-free projection of A and let f be a normalized probability over the set of atomic graph edits. Then, we define the

probability of observing G given A as:

$$p_O(G|A) = p_f(G|G_A) = \max_{D \in \mathbf{D}_{(G_A,G)}} \prod_{d \in D} f(d) \tag{13.1}$$

where $\mathbf{D}_{(G_A,G)}$ is the set of all possible edits that convert G_A to G. In other words, the likelihood of G given A is a function of how different G is from the noise-free projection of A. Here, we assume that edits are mutually independent and that $V_{\text{alt}} = V(G) \setminus V(G_A)$. As noted above, estimating the minimum edit distance between two arbitrary graphs is NP-hard. However, we will now define a noisy graph search algorithm that will allow us to efficiently estimate (13.1) for a large set of graphs similar to the initial noisy graph.

13.1.4 Noisy graph search

Our strategy for addressing potential errors in the input planar graph G is to search the embedded meta-digraph starting at vertex G using a similar algorithm to our heuristic search algorithm defined in Chapter 10. Here, we assume that the set of alternate vertices V_{alt} is given. In practice, it can be estimated by a key-point selection algorithm such as the one described in Appendix C.

In more detail, given a noisy graph G, a set of alternate vertices V_{alt}, and a growth model M, we first estimate the greedy flow-dag as defined in Chapter 10:

$$G_d = \texttt{greedy\_search}(G, M)$$

for the input graph G. We then refine the greedy solution using our heuristic search algorithm:

$$G_d = \texttt{heuristic\_directed\_tree\_search}(G, G_d, M).$$

Afterwards, we enumerate all possible single atomic edits D that could be applied to G given V_{alt} and determine the probability of each edit using the cost function f:

$$D = \texttt{possible\_atomic\_edits}(G, V_{\text{alt}}, f).$$

We then apply each edit d in turn to obtain a new graph and a new set of alternate vertices:

$$[G', V'_{\text{alt}}] = \texttt{edit\_graph}(G, d).$$

We test to see if the edited graph G' is still planar:

$$\texttt{is\_planar}(G')$$

which can be done in logarithmic time if the initial graph was planar (Di Battista and Tamassia, 1996). Finally, we estimate the greedy solution for G'. Our heuristic function is a convex combination of the probability of the greedy solution and the probability of the proposed edit:

$$h = \lambda \ell_M(G'_d) + (1 - \lambda)\ell_f(d),$$

where $\ell_f(d)$ is the log-probability of the edit given f and $\lambda \in [0, 1]$. For each graph G that we pop from the queue, we estimate the posterior log-probability of the most likely directed tree T for G, relative to the initial graph G_0:

$$\ell(T) = \ell_f(D(G_0, G)) + \ell_M(T),$$

where $\ell_f(D(G_0, G))$ is the joint log-likelihood of the edits that converted G_0 to G and $\ell_M(T)$ is the prior log-probability of the directed tree. In our algorithm, each graph in the queue keeps tracks of the edits that lead to it, so determining $D(G_0, G)$ is trivial. Our proposed noisy graph search algorithm is summarized in Algorithm 3.

The main challenge to bring this extension to fruition is how to deal with the vastly larger search space. By searching over both planar graphs and directed trees, the number of possible solutions becomes doubly exponential. Clearly, unless our heuristics are very accurate, this search space will be unmanageable.

Algorithm 3: Noisy graph directed tree search

Input: Noisy planar graph G, alternative vertices V_{alt}, edit cost function f, growth model M, max priority queue q, max number of iterations $i_{\max} > 1$

Output: Locally optimal directed tree $T^\star$.

```
G_0 = G;
G*_d = greedy_search(G,M);
ℓ* = ℓ_M(G*_d);
q = push(G,G*_d,V_alt,ℓ_M(G*_d));
for i ← 1 to i_max do
    [G,G_d,V_alt] = pop(q);
    G_d = heuristic_directed_tree_search(G,G_d,M);
    if ℓ* < ℓ_M(G_d) then
        ℓ* = ℓ_M(G_d) + ℓ_f(D(G_0,G));
        G*_d = G_d;
    D_s = possible_atomic_edits(G,V_alt);
    foreach d ∈ D_s do
        [G',V'_alt] = edit_graph(G,d);
        if not_visited(G') & is_planar(G') then
            G'_d = greedy_search(G',M);
            h = λℓ_M(G'_d) + (1 − λ)ℓ_f(d);
            q = push(G',G'_d,V'_alt,h);
T* = G*_d;
foreach v ∈ T* do
    if deg⁻(v) > 1 then
        (ℓ,P) = partition(v,T*,M);
        T* = transform(T*,P);
```

13.2 Robust graph extraction

In the previous section, we proposed an algorithm that explores not only different directed trees given a planar graph, but also modifies the input graph so as to obtain an overall higher probability solution. However, the number of possible edits for a given graph grows exponentially with the number of vertices. Therefore, the above scheme will only be practical if the initial noisy graph is close to the correct planar graph.

In this section, we outline two potential improvements to our current method that may significantly improve the graph extraction process. The first change is to make use of our tree growth model to constrain the extraction; the second is to dynamically estimate the most likely tree as we extract the planar graph.

13.2.1 Growth-based track estimation

In our current graph extraction method we first obtain a series of tracks that attempt to follow the branches of the tree in the image. We use a combination of shape, orientation and radiometric features to ensure that each track follows a branch-like path in the image. However, each track is grown independently of other tracks. Therefore, we have no *a priori* constraints on the overall shape of the resulting graph.

To address this shortcoming, here we propose instead to grow the set of tracks, and hence the planar graph, in a manner analogous to the tree growth model presented in Chapter 7. In this approach, we grow all the tracks that span the tree's branches simultaneously. At each iteration, the end-point segment of each track either stops growing or spawns a set of c child segments based on the local image properties.

In more detail, at iteration $t = 0$, we start from a segment centered at the tree's root. Then, at each iteration $t > 0$ we keep a frontier F of tracks that are still growing. At each iteration, each end-point segment in $F(t)$ spawns a number $c \geqslant 0$ of child segments. Let C be a set of possible children of the frontier segment S. Then, the joint probability of C is given by:

$$p(C|S) = p_s(c) \prod_{S_i \in C} p_a(S_i|S)$$

where S_i is the i-th child of S, $p_s(c)$ is defined as in (7.1) and $p_a(S_i|S)$ is a function of both the angle between the expected direction of growth for the i-th child and the

local image properties at S_i:

$$p_a(S_i|S) \propto \text{SSD}\,(S, S_i'), \tag{13.2}$$

where SSD denotes the sum-of-squared differences between two segments and S_i' is the segment weighed by a von Mises kernel as defined in Appendix C:

$$S_i'(\mathbf{p}) = S_i(\mathbf{p})\text{VM}(\mathbf{p} - \mathbf{c}; \boldsymbol{\mu}_i, \kappa),$$

where $\mathbf{p}$ lies inside S_i. Here $\mathbf{c}$ is the center of segment S and $\boldsymbol{\mu}_i$ is the expected direction of growth for the i-th child of S.

As with the growth model of Chapter 7, the segment S is then removed from the frontier and any child segments it spawned are added to $F(t+1)$.

The proposed method differs from our current approach in three key ways:

1. A segment may have more than one child.

2. All tracks are grown simultaneously.

3. New tracks spawn from existing tracks.

By growing the planar graph as a tree, we hope to more closely estimate the branches, branch-points and branch-crossings of the projected tree.

13.2.2 On-line tree estimation

A further potential improvement on the graph extraction process outlined above is to simultaneously estimate the most likely tree as the planar graph is being extracted. In other words, at iteration t we determine the most likely tree for the planar graph obtained until that point. Then, at iteration $t+1$, we estimate how the probability of the most likely tree varies if we add each of the proposed segments. If a segment significantly affects the tree's probability, it is discarded and not added to the planar graph. Therefore, in this approach each new segment is weighted not only by how

well it fits with its track, but how it modifies the overall probability of the most likely tree given the current set of tracks.

In this scheme, the decision of whether or not to add a segment to a track is determined not only by local, but also by global constraints, which should help regularize the graph extraction process and lead to a more accurate estimate of the original tree's topology.

In order to make the above estimation tractable we need to be able to quickly determine the most likely tree at each iteration. Two potential solutions are:

1. Use only the greedy solution.

2. Develop an online update algorithm that will efficiently update the most likely tree at each iteration.

An online update algorithm is an algorithm that can efficiently determine–sometimes in logarithmic time–the best solution at iteration $t + 1$, given the best solution at iteration t. For instance, the fastest algorithm for the classic assignment problem runs in $O(n^3)$ (Kuhn, 1955) for a static graph; however, if new vertices and edges are added or removed from the graph, it is possible to update the optimal assignment in only $O(n^2)$ (Toroslu and Üçoluk, 2007).

The first solution consists of estimating the most likely tree at each iteration using our linear greedy search algorithm. The second solution is clearly more desirable, but at this point it is not clear whether or not an online updating scheme is possible for the tree estimation problem.

13.3 Noise Sampling

In the previous sections, we outlined two ways of addressing the fact that state-of-the-art graph extraction methods are not yet robust enough for many applications. In this section, we will explore a different potential improvement to our tree estimation

methodology. We propose a novel sampling technique that will allow us to obtain different, but still likely greedy solutions. These solutions may lie arbitrarily far from each other in the meta-graph, thereby lessening our dependence on the quality of the initial greedy solution.

Our tree estimation approach consists of two key steps: first we estimate a greedy tree and then we refine this greedy solution using a heuristic search that explores a local neighborhood around the greedy tree. As noted in Chapter 6, the number of possible solutions grows exponentially with the number of vertices in the graph. Therefore, as the input graph gets larger, the quality of our final estimated tree becomes more dependent on the quality of the greedy solution.

Although we showed in Chapter 12 that our greedy algorithm accurately approximates the correct solution for various types of trees, including retinal vessels and plant roots, we would like to be able to converge to a good solution even if the initial greedy guess was poor.

In stochastic optimization terms, our heuristic search only makes *local* changes to the greedy solution in the meta-graph. If the initial guess is far from the true optimum, it will take a long time to converge to the correct value. To address this limitation, here we propose *noise sampling*, a novel sampling technique that allows us to obtain different likely initial solutions. As we will see, any two samples may lie arbitrarily far from each other in the meta-graph, thereby allowing us to apply *non-local* changes to our solution. By performing several searches starting at different promising locations in the meta-graph, we are more likely to converge on the true optimum.

We will first define noise sampling in a general setting and then apply it specifically to sample greedy trees.

13.3.1 Problem definition

Variational approximation and sampling are two commonly used stochastic strategies for optimizing an NP-hard problem. In the former, instead of solving the exact problem, we solve a similar, but more tractable problem, which is often obtained by computing a bound on the values of interest. In the latter approach, we explore the space of possible solutions by obtaining a sequence of random samples from a distribution which is proportional to the value of each solution.

The benefits and drawbacks of each approach are well known: variational methods are generally fast, but intrinsically inaccurate, while sampling methods are slow, but have optimality guarantees in the limit as the sample size grows to infinity. Here, we develop a hybrid sampling technique that allows us to combine several of the benefits–while avoiding many of the drawbacks–of both approaches.

Our method is particularly well suited for combinatorial problems, such as those that arise in graph theory, in which a large set of possible solutions is comprised of different combinations of elements of a much smaller set.

Let Ω and $\mathcal{X}$ be a pair of discrete sets such that:

$$\mathcal{X} \subseteq \mathcal{P}(\Omega),$$

where $\mathcal{P}$ denotes the power set of Ω. In other words, each element $\mathbf{x} \in \mathcal{X}$ is comprised of a set of elements from Ω:

$$\mathbf{x} = \{\omega_1, \omega_2, \cdots, \omega_m\},$$

for $\omega_{1:m} \in \Omega$. Now, let Q be a likelihood function defined over $\mathcal{X}$. If Q is normalized, such that $Q : \mathcal{X} \mapsto [0, 1]$ and $Q(\mathcal{X}) = 1$, then Q is a probability distribution.

Here, we assume that $\mathcal{X}$ corresponds to the possible solutions of an NP-hard optimization problem and that Q is a (potentially unnormalized) *approximate* likelihood over these solutions. We also assume that maximizing Q is tractable; that is, the

value

$$\mathbf{m}_Q = \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}} Q(\mathbf{x})$$

can be obtained in polynomial time. Furthemore, we assume that each value $Q(\mathbf{x})$ is a function of the individual values of the elements of $\mathbf{x}$. More concretely, let $\phi(\omega) : \Omega \mapsto \mathbb{R}$ be a function defined over the elements of Ω. This function will often correspond to the individual likelihood of each ω. We then denote Q as a *combinatorial function* if it is a function of these values:

$$Q(\mathbf{x}) = f(\phi(\omega_1), \phi(\omega_2), \cdots, \phi(\omega_m)).$$

In other words, the set $\{\phi_1, \phi_2, \cdots, \phi_{|\Omega|}\}$ constitutes a basis for Q. The simplest case is when the basis is linear:

$$Q(\mathbf{x}) = \mathbf{w}_{\mathbf{x}}^{\top} \boldsymbol{\phi},$$

where

$$\mathbf{w}_{\mathbf{x}} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{|\Omega|} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\phi} = \begin{bmatrix} \phi(\omega_1) \\ \phi(\omega_2) \\ \vdots \\ \phi(\omega_{|\Omega|}) \end{bmatrix}.$$

Here, elements of Ω which are not part of $\mathbf{x}$ are ignored:

$$\omega \notin \mathbf{x} \rightarrow w(\omega) = 0.$$

Finally, we denote Q as unweighted when all the non-zero weights are unitary:

$$\omega \in \mathbf{x} \rightarrow w(\omega) = 1.$$

Thus, in the linear case we simply add the (potentially weighted) values of each ω to obtain the global value of each $\mathbf{x}$.

A number of combinatorial problems fall into this framework. For instance, let $G = (V, E)$ be a weighted graph. We can formulate finding the shortest path between

two vertices as follows. Let $\Omega = E$, $\mathcal{X}$ be the set of paths, or connected edges in G, and let $Q(\mathbf{x})$ be the sum of the edges of each path in $\mathcal{X}$. Similarly, the minimum spanning tree problem consists of exploring a combinatorial space where $\Omega = E$, $\mathcal{X}$ is the set of spanning trees over G and Q assigns the cost of each tree as the sum of the cost of its edges.

More generally, the ϕ basis functions can be combined in any way to form the values of Q. As we will see, however, all we require for noise sampling to be effective is for it to be tractable to optimize Q.

Now, let $\mathbf{m}^\star$ be a true maximum of the NP-hard problem:

$$\mathbf{m}^\star = \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}} Q^\star(\mathbf{x}).$$

where $Q^\star$ is the true likelihood over $\mathcal{X}$ for the optimization problem at hand.[1] Since we can determine $\mathbf{m}_Q$ efficiently, this value can be used as an initial approximation for $\mathbf{m}^\star$, which we can then refine using local search or MCMC sampling. However, in general, the fact that

$$\|Q^\star(\mathbf{m}^\star) - Q^\star(\mathbf{m}_Q)\|_2$$

is small is no guarantee that

$$\|\mathbf{m}^\star - \mathbf{m}_Q\|_d$$

is also small, where $\|\|_d$ is a suitable metric over $\mathcal{X}$. In fact, the above distance may be arbitrarily large if $Q^\star$ is multi-modal. In this case, local refinement techniques will fail to significantly improve the estimated solution. To address this limitation, we will now present a way to obtain samples from $\mathcal{X}$ which are likely to have high values in Q.

[1] For simplicity, we assume, w.l.o.g., that $\mathbf{m}^\star$ is unique.

13.3.2 Noisy maximization

Let $\hat{Q}$ be a noisy version of Q, such that:

$$\hat{Q}(\mathbf{x}) = N_{\mathbf{x}}(Q(\mathbf{x}); \theta_{\mathbf{x}}), \quad \forall \mathbf{x} \in \mathcal{X}, \tag{13.3}$$

where $N_{\mathbf{x}}$ is a noise distribution that depends on the original value of $Q(\mathbf{x})$ and $\theta_{\mathbf{x}}$ are any additional distribution parameters. For instance, $N_{\mathbf{x}}$ can correspond to a Gaussian distribution with mean $Q(\mathbf{x})$ and variance $\theta_{\mathbf{x}} = \sigma^2$.

As before, let

$$\mathbf{m}_{\hat{Q}} = \operatorname*{argmax}_{\mathbf{x} \in \mathcal{X}}(\hat{Q}(\mathbf{x}))$$

be the value that maximizes $\hat{Q}$. We denote obtaining $\mathbf{m}_{\hat{Q}}$ as a *noisy maximization* (NM) of Q. For many combinatorial problems, such as graph matchings and spanning trees, determining $\mathbf{m}_{\hat{Q}}$ is as tractable as determining $\mathbf{m}_Q$, since the exact shape of Q is a not a factor in obtaining its maximum. In general, for noise sampling to be tractable, we simply require that obtaining $\mathbf{m}_{\hat{Q}}$ be tractable.

More importantly, $\mathbf{m}_Q$ need not be the same as $\mathbf{m}_{\hat{Q}}$ since the noise may push a second, potentially distant, element of $\mathcal{X}$ to have a maximum value in $\hat{Q}$. Intuitively, given two independent noisy likelihoods $\hat{Q}_1$ and $\hat{Q}_2$, we expect that the elements $\mathbf{m}_{\hat{Q}_1}$ and $\mathbf{m}_{\hat{Q}_2}$ may be different and, if Q is multi-modal, these two values may lie arbitrarily far from each other in $\mathcal{X}$.

Clearly, each value produced by noisy maximization is a sample from $\mathcal{X}$. We will now show that if Q is a combinatorial function and the noise satisfies certain properties, then NM can be used as a proposal distribution that spans all of $\mathcal{X}$.

More concretely, let $R(\mathbf{x})$ be the probability that $\mathbf{m}_{\hat{Q}} = \mathbf{x}$. In order to be able to use NM to sample from $\mathcal{X}$, we require that R be *exhaustive*. That is:

$$R(\mathbf{x}) > 0 \quad \forall \mathbf{x} \in \mathcal{X}. \tag{13.4}$$

Thus, it must be possible for any element in $\mathcal{X}$ to be the maximum of some realization of $\hat{Q}$.

We will now show that if Q is a combinatorial function and the noise function is of the form:

$$\hat{\phi}(\omega) = \mathcal{N}(\phi(\omega), \sigma^2),$$

where (N) is a Gaussian distribution with variance σ^2, then $R(\mathbf{x})$ is exhaustive.

Proof. The value of every $Q(\mathbf{x})$ is a function of the values of each of the elements of $\mathbf{x}$:

$$Q(\mathbf{x}) = f(\phi(\omega_1), \phi(\omega_2), \cdots, \phi(\omega_m)).$$

After noise maximization, the value $\hat{Q}(\mathbf{x})$ is given by:

$$\hat{Q}(\mathbf{x}) = f(\hat{\phi}(\omega_1), \hat{\phi}(\omega_2), \cdots, \hat{\phi}(\omega_m)).$$

Since the Gaussian distribution has infinite support, there is a non-zero probability for each $\hat{\phi}(\omega_i)$ to take on any value in $\mathbb{R}$ and hence for $\hat{Q}(\mathbf{x})$ to also take on any value in the range of f. Therefore, there is a non-zero probability that $\hat{Q}(\mathbf{x})$ will be the maximum value of $\hat{Q}$. □

Furthermore, in the above case NM is *order-preserving*, in the sense that if $Q(\mathbf{x}) > Q(\mathbf{y})$, then $R(\mathbf{x}) > R(\mathbf{y})$; that is, it is more likely for the noisy maximum to correspond to a high value of Q than to a low one. We will first show what property R needs to satisfy to be order-preserving and then show that it is satisfied in the Gaussian noise case.

Let $\mathbf{x}$ and $\mathbf{y}$ be two elements of $\mathcal{X}$ such that $Q(\mathbf{x}) > Q(\mathbf{y})$ and let $N_{\mathbf{x}}$ and $N_{\mathbf{y}}$ be their respective noise distributions. Noisy maximization is order-preserving if:

$$R(\mathbf{x}) > R(\mathbf{y}). \tag{13.5}$$

Now, let X and Y be the random variables which are distributed according to $N_{\mathbf{x}}$ and $N_{\mathbf{y}}$, respectively. Then, (13.5) holds if and only if:

$$Q(X > Y) > Q(X < Y). \tag{13.6}$$

The above equation simply states that it is more likely for X to be greater than Y than viceversa. Now, let $Z = X - Y$ be the difference distribution of these two random variables. Equation 13.6 can be rewritten as:

$$Q(Z > 0) > Q(Z < 0). \tag{13.7}$$

The above equation is true if and only if the *median* of Z is positive.

Proof. By definition, for any random variable, the median splits its cumulative distribution in two equal parts:

$$Q(Z > \text{median}\,(Z)) = Q(Z < \text{median}\,(Z)) = 1/2.$$

In order for (13.7) to be true, we require that $Q(Z < 0) < 1/2$, so:

$$Q(Z < 0) < Q(Z < \text{median}\,(Z)).$$

Since the cumulative distribution is monotonic, the above equation is only true if:

$$\text{median}\,(Z) > 0.$$

□

Thus, R is order-preserving if and only if:

$$Q(\mathbf{x}) > Q(\mathbf{y}) \rightarrow \text{median}\,(N_{\mathbf{x}} - N_{\mathbf{y}}) > 0.$$

We will now show that the above property holds when Q is a combinatorial function and the noise function is symmetric (e.g. Gaussian).

Proof. Let $\mathbf{x}$ and $\mathbf{y}$ be two elements of $\mathcal{X}$, such that $Q(\mathbf{x}) > Q(\mathbf{y})$. Now, let X and Y be the two random variables for their corresponding symmetric noise distributions and let $Z = X - Y$ be their difference distribution. The mean of Z is given by:

$$\mu_Z = \mu_X - \mu_Y.$$

The difference distribution of two symmetric distributions is also symmetric, so:

$$\text{median}\,(Z) = \mu_Z.$$

Since $\mu_X > \mu_Y$, then $\text{median}\,(Z) > 0$. □

In summary, it is clear that by adding noise to the values of ϕ, we implicitly add noise to the values of Q. This indirect noise addition allows us to apply NM to sets of possible solutions which are too large to enumerate directly. Furthermore, under the conditions listed in the above proof, NM can be used as a proposal distribution that spans all of $\mathcal{X}$, as detailed further in the next section.

13.3.3 MCMC-NM sampling

In the previous section, we showed that after adding suitable noise to the values of a distribution, the new maximum value could potentially correspond to any value in the distribution's support but is more likely to correspond to a high value than to a low one when the noise distributions are symmetric. Here, we will show that we can use a sequence of noisy maximization values to efficiently sample from the original distribution using Markov Chain Monte Carlo (MCMC).

We will show how to make use of noisy maximization (NM) in a Metropolis-Hastings MCMC scheme (more specifically, a Metropolis algorithm, since NM is symmetric). It is also possible, however, to incorporate NM into other sampling techniques, such as Gibbs sampling or rejection sampling.

The Metropolis algorithm works as follows. Given a current value $\mathbf{x}_t$, it generates a new value $\mathbf{x}_{t+1}$ from a distribution Q by accepting or rejecting a value drawn from a proposal distribution Q'. The main steps are:

1. Propose a new sample value $\mathbf{x}'$ from the proposal distribution $Q'(\mathbf{x}'|\mathbf{x}_t)$.

2. Calculate the ratio $a = \frac{Q(\mathbf{x}')}{Q(\mathbf{x}_t)}$.

3. If $a \geqslant 1$, accept $\mathbf{x}'$, such that $\mathbf{x}_{t+1} = \mathbf{x}'$.

4. Else, make $\mathbf{x}_{t+1} = \mathbf{x}'$ with probability a or make $\mathbf{x}_{t+1} = \mathbf{x}_t$ with probability $1 - a$.

Provided the proposal distribution satisfies certain basic conditions, the Metropolis sampling scheme described above will sample from the desired, underlying distribution (Bishop, 2006).

In order to use NM in an MCMC scheme, we draw the new proposed sample $\mathbf{x}'$ as follows:

1. Obtain a new noisy likelihood: $\hat{Q}(\mathbf{x}) = N_{\mathbf{x}}(Q(\mathbf{x}); \theta_{\mathbf{x}})$.

2. Find its maximum: $\mathbf{m}_{\hat{Q}} = \operatorname{argmax} \hat{Q}(\mathbf{x})$.

3. Propose this maximum: $\mathbf{x}' = \mathbf{m}_{\hat{Q}}$.

As a proposal distribution, noisy maximization has a number of desirable properties. As noted before, its support covers all of the underlying space and the probability that a proposal value will be chosen is monotonically proportional to its value in the base distribution. Also, NM does not make use of an additional metric between the values of $\mathcal{X}$. This means that we do not require an index over $\mathcal{X}$ to be able to explore it.

Most importantly, NM is a global, not a local, way of selecting a proposal value. Thus, one key difference between MCMC-NM and MCMC with a standard Gaussian proposal distribution is that, if the noise added at each iteration is i.i.d., then the NM samples are less correlated. More concretely, let $R_{t+1}(\mathbf{x}|\mathbf{x}_t)$ be the probability that x will be the proposed new sample at time $t+1$, given that the previous sample was $\mathbf{x}_t$. Then,

$$R_{t+1}(\mathbf{x}) = R_{t+1}(\mathbf{x}|\mathbf{x}_t), \quad \forall \mathbf{x} \in \mathcal{X}.$$

Proof. The noise added at each timestep is i.i.d. and independent of the current sample. Therefore, the location of $\mathbf{x}_t$ has no impact on the maximum value of $\hat{Q}$ at time $t+1$. □

Thus, the only correlation between values of the Markov chain occurs when the proposed sample is rejected, in which case $\mathbf{x}_{t+1} = \mathbf{x}_t$ and thus the two samples are correlated. In effect, there is no need to have a burn-in period in MCMC-NM, since the chain will forget its starting location as soon as the first new sample is accepted.

13.3.4 Greedy tree sampling

We are now ready to apply our proposed noise sampling to our tree estimation case. As defined in Chapter 10, given an undirected graph G with root r, our greedy algorithm estimates a flow-dag based on a shortest-path spanning tree of G. Therefore, in this case $\Omega = E(G)$ and $\mathcal{X}$ is the set of all possible spanning trees over G. Then, $\phi(\omega)$ is the cost of edge ω and $Q(\mathbf{x})$ is the sum of the costs of the distances from r to every other vertex in the spanning tree $\mathbf{x}$.

Clearly, Q is a combinatorial function. Therefore, we can easily different greedy solutions as follows:

- Add Gaussian noise to the cost of each edge.
- Estimate a shortest-path tree.
- Orient the remaining edges of G according to a topological ordering defined by the shortest-path tree.
- Accept or reject the proposed greedy solution using the Metropolis criterion.

For each greedy solution that we accept, we can then refine this solution using our heuristic directed tree algorithm. Since the proposed greedy solutions need not be close to each in the meta-graph–that is, they do not have to agree on most of their

edge orientations–then combining MCMC-NM sampling with our tree estimation algorithms should allow us to more effectively estimate the most likely tree.

The main concern that we need to address in order to fully incorporate this sampling technique into our estimation methodology is how to avoid redundancy in our heuristic search. That is, if we run two heuristic searches based on two greedy graphs which are close to each other, then we will likely visit many of the same solutions in both searches. We thus need to develop ways to minimize this overlap when searching in parallel.

13.4 Branch ordering and AV classification

In the previous sections, we outlined a number of improvements to our tree estimation algorithms that may allow us to better estimate the most likely tree. In this section, we will explore two extensions to our methodology which are relevant to various retinal vessel applications. We will first consider how to estimate the original ordering of the branches; that is which branches are closer to the plane and which are further away. Determining which vessels are deeper in the retina and which are closer to the surface will give us further insight into their anatomical function. Finally, we will consider the problem of classifying arteries vs. veins. As noted in Chapter 2, the cardiovascular system is subdivided into two main subtrees, one which carries blood out of the heart and another into it. Here, we will sketch out some possible ways of incorporating this domain knowledge into our tree estimation, in order to better distinguish these two types of vessels in the retina.

13.4.1 Branch ordering

In this work, given a projected graph, we have only concerned ourselves with determining a tree whose projection is isomorphic to the original graph. However, in some cases we may be interested not only in determining the original tree's topology,

but also in estimating the ordering of its branches relative to the plane of projection. In other words, in this case we are interested in determining which branches in the image are closer and which are further away. For instance, the retina is made up of a set of distinct layers and each subtree tends to be part of the same layer. Determining which vessels lie at which layer can give us more insight into their function in the eye.

Formally, let T be an estimated tree and let $\alpha(V(T)) = \{V_1, V_2, \cdots, V_n\}$ be the partition of the vertices of $V(T)$ such that the vertices in each subset project to the same location on the plane:

$$\{u, v \in V_i \Leftrightarrow P(u) = P(v), \quad \forall u, v \in V(T)\}.$$

We then seek a partial ordering of the tree by defining a total ordering on each V_i:

$$\{\forall u, v \in V_i \mid u < v \quad \text{or} \quad v < u\}.$$

Let $A, B \subset E(T)$ be two branches of T which cross each other in the projection. Then, given a set of total orderings on the crossings, we can further classify the relationship between the two branches:

1. A is always closer than B.
2. B is always closer than A.
3. A and B wrap around each other.

Being able to distiguish between these three cases gives us further insight into the original geometry of T.

However, defining a total ordering at each crossing is a challenging problem. First, when a set of points are arranged in a projective line, the closest point occludes the remaining points. Therefore, even in cases in which we have enough visual information to determine the closest point to the plane, it is not possible in general

to determine the relative order of the points which are further away. Furthermore, sometimes even determining the closest point may be challenging due to lack of visual cues, as the lower half of Figure 13.1 illustrates. We will now outline some potential approaches to this problem.

Branch ordering estimation

Given a projected graph G and an estimated tree topology T, we wish to impose a total ordering on each set of co-projecting vertices of T. As Figure 13.1 shows, sometimes we can easily determine this ordering due to visual cues at the branch-crossing itself. For instance, if one branch is lighter than the other and the branch-crossing is also light, we can deduce that the light branch is closest and vice-versa. However, the same figure also highlights cases in which there is not enough visual information to clearly tell which branches obscure which.

Generally, the larger, thicker branches in an image will be easier to order while the smaller branches will be more ambiguous. Nevertheless, in some types of trees, such as retinal vessels, if one branch is closer than another at point $\mathbf{p}$, is is more likely to also be closer at a second point $\mathbf{p}'$. In other words, in some types of trees, branches do not twist or wrap around each other; rather, the relative ordering of branches is fairly constant.

One possible strategy for ordering the branches under this assumption is as follows. We will first identify a set of easily discernible cases, such as those of Figure 13.1 (a,b,c) and then use the known ordering at those locations to disambiguate the harder cases, such as those of Figure 13.1 (d,e,f).

A potential approach is to formulate the overall ordering of the directed tree T as a weighed constraint satisfaction problem (Larrosa and Schiex, 2004) in which the ambiguous crossings are free variables and the easy crossings constrain or bias the possible solutions at the ambiguous crossings.

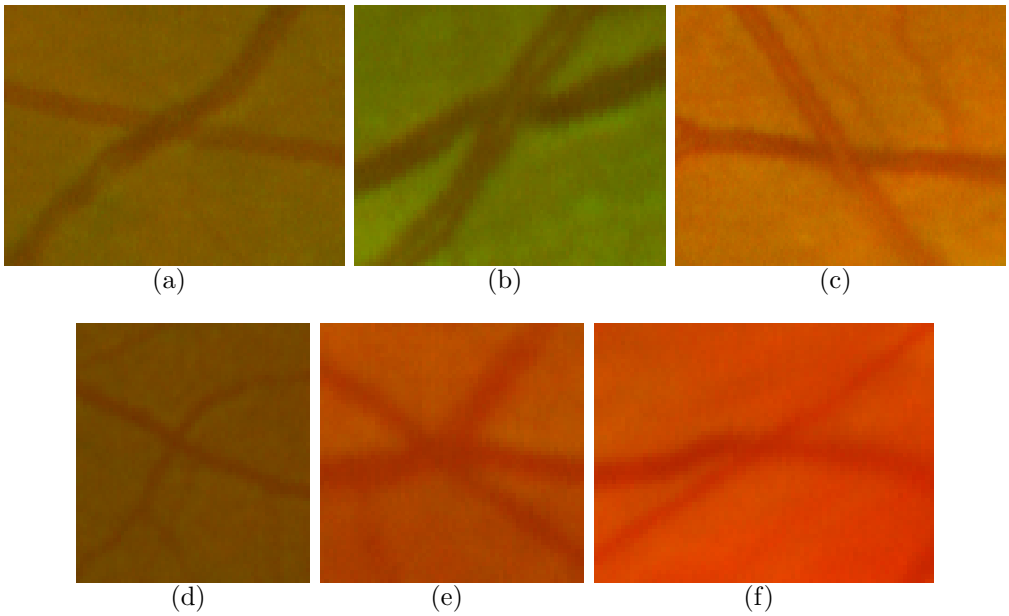


FIGURE 13.1: **Ordered branch crossings:** Six examples of branch-crossings taken from different retinal vessel images. In **(a,b,c)**, there is enough visual information at the branch-crossing itself to ascertain which branch is closest. In **(d,e,f)**, on the other hand, the image is much more ambiguous and the relative ordering of each branch is not clear.

13.4.2 Artery vs. vein classification

A direct extension of our tree estimation framework is to apply it to the problem of distinguishing arteries from veins (Perez et al., 2002; Rothaus et al., 2007; Dashtbozorg et al., 2013). The various vessels in a retinal image actually correspond to two different classes; the arteries push oxygenated blood out of the optic nerve into the retina and the veins draw the deoxygenated blood back towards the heart.

The tree growth model and the tree estimation algorithms presented in this work are general, in the sense that they make few assumptions about the global properties of the tree. However, the fact that the vessels in the retina are subdivided into two subtrees that emerge from the root at the optic nerve provides additional domain knowledge that we can leverage to optimize our tree search.

As Figure 13.2 illustrates, the subtrees corresponding to the arteries and vein in the retina display two key features:

1. The two subtrees have similar numbers of branches.

2. Arteries and veins cross each other far more than they self-cross.

Given these additional features about retinal vessels, it should be possible to further refine our tree estimation algorithms so that they better estimate the most likely sets or arteries and veins. For instance, we could measure the relative size of the two main subtrees in the tree and compare how much subtree overlaps with the other as opposed to itself.

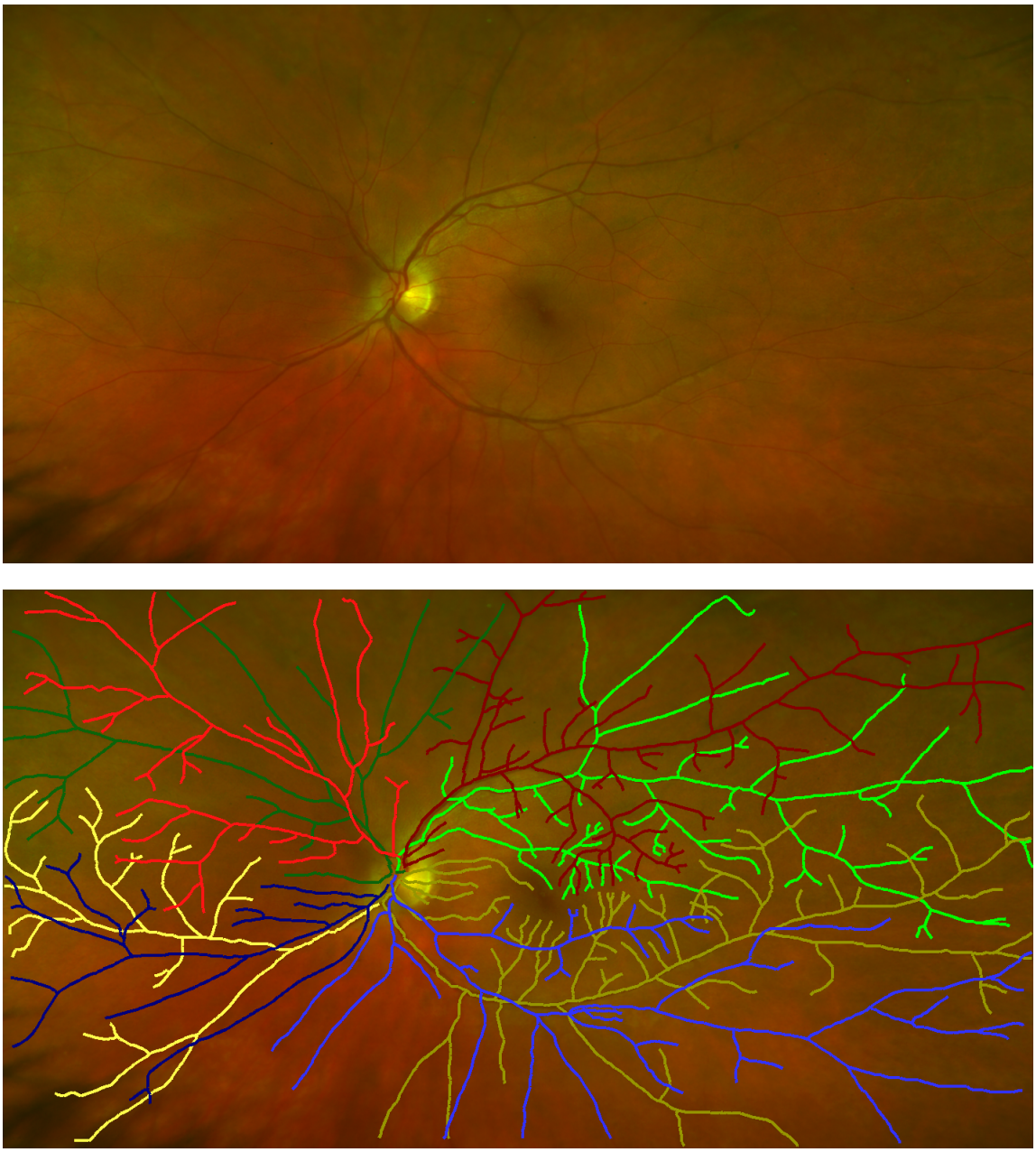


FIGURE 13.2: **Arteries vs. Veins:** In a retinal image, the vessels are divided into either arteries or veins. **(a)** A retinal image. **(b)** The ground truth tree. The red and yellow subtrees correspond to veins while the green and blue ones correspond to arteries. Different shades represent different sub-subtrees. Note how arteries and veins cross each other far more than they self-cross.

14

Conclusions

Trees are ubiquitous physical structures that efficiently distribute a flow between a central source and a set of end-points. However, the three-dimensional topology of a tree is lost in a two-dimensional image of it. In this work, we have formalized the problem of estimating the topology of a three-dimensional tree from a single, two-dimensional image. We showed that projecting a tree obscures both its original topology and the directions of its edges and then characterized the set of possible source trees given a projected graph in terms of edge orientations and vertex partitions.

Tree growth models help to regularize this estimation problem. However, we also showed that for certain classes of growth models this problem is NP-hard. We thus presented a heuristic search method that allows to efficiently explore the space of possible trees. Our experiments in Chapter 12 empirically showed that our proposed methodology effectively approximates the most likely solution for various types of trees, including retinal vessels and plant roots.

Our exploring of this topic still leaves many avenues for further work, including developing a more robust graph extraction method, using sampling to improve

our heuristic search and making use of domain knowledge to further regularize our solution.

Additional possibilities include analyzing images that include incomplete or partially occluded trees and exploring additional image features such as branch thickness and color to further disambiguate between different possible trees. Another intriguing possibility is to extend our methodology to other tubular structures, such as knots, biological neural networks, and capillaries, which may have circuits and hence not be trees. We look forward to exploring these exciting new research directions.

Appendix A

Image Pre-Processing

Our overall goal is to estimate the most likely topology of a tree given an image of it. Generally, however, trees do not exist in a vacuum; the raw image of a tree includes a number of extraneous elements, including nearby objects, imaging artifacts and other aberrations that obscure the actual tree. Before we attempt to estimate the tree's topology, we first have to determine which parts of the images actually correspond to the tree in question.

In general, isolating the pixels that correspond to the tree from the rest of an image consists of three steps:

1. Artifact & extraneous object removal
2. Branch enhancement
3. Tree segmentation

In the first step, we identify and remove any imaging artifacts, such as lens flares, false colors, or aliasing, that are a by-product of the imaging process, as well as regions of the image that cannot include any tree pixels. In the second step, we determine,

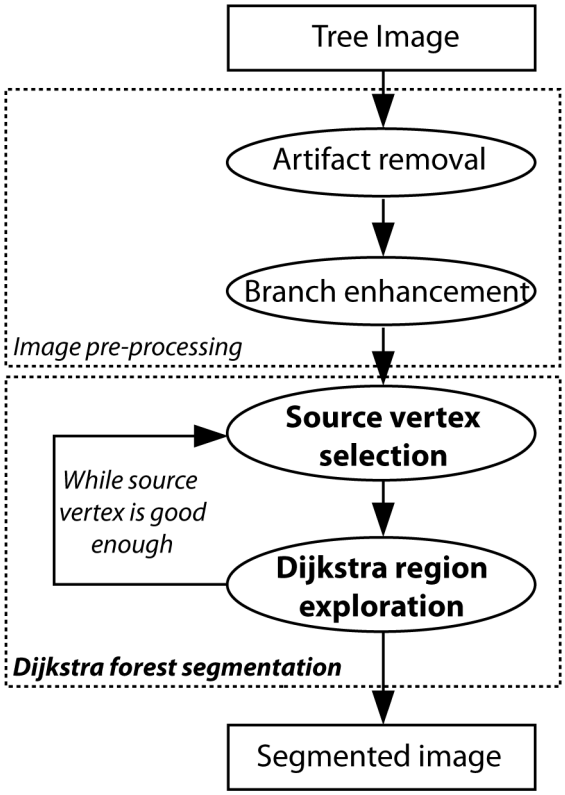


FIGURE A.1: **Tree segmentation pipeline:** First, we apply directional local-contrast filters (DLCF) and LoG-Gabor filters to eliminate artifacts and increase contrast. Then, the best, unvisited tree pixel in the image is repeatedly chosen as a starting point for a dynamic-programming exploration of the unvisited part of the image. The result of each exploration yields a new tree in the growing forest. Forest growth stops when the best, unvisited pixel is worse than a predefined threshold.

for each pixel, the likelihood that it is part of the tree. In the last step, we apply a thresholding function to this likelihood to partition the image into tree and non-tree pixels. Figure A.1 illustrates our complete tree segmentation pipeline. In this appendix, we will address the first two steps; we will then tackle image segmentation in the next appendix.

A.1 Artifact and extraneous object removal

In general, trees do not exist in a vacuum. When we image a tree, we also image its surroundings, which often include nearby objects, such as other trees, the

medium on which the tree is growing, or even parts of our imaging device. Furthermore, many trees of interest require specialized imaging devices; for instance, retinal vessels require fundus imaging while lung airways require CAT scans. The images produced by these devices often include lens and compression artifacts, which have to be addressed in order to correctly extract the imaged tree.

In order to analyze a tree, we have to be able to isolate it from the rest of the image. While the exact procedure is highly domain-dependent, in this section we present a pipeline which has proven effective for analyzing retinal vessels in color fundus images (Estrada et al., 2011) and which can serve as a template for how to pre-process other types of tree images.

Figures A.2 and A.3 illustrate two different types of fundus images. In this domain, the main challenges consists of removing lens artifacts which arise from the imaging process and isolating the retina, (where the vessels are located) from the rest of the image (where they are not). To address artifact removal, we apply directional local contrast filtering to remove high-saliency lens artifacts. We then isolate the retina, the region on which all the vessels of interest are found, from the rest of the image using hue-saturation-value classification.

We view each retinal color image as an $N \times M \times 3$ matrix I. A pixel position in I is given by a two-dimensional vector of integers, $\mathbf{p} = [x, y]^\top$. The value at each pixel position is given by a three-dimensional vector $I(\mathbf{p})$ of red, green, blue values normalized between 0 and 1.

A.1.1 DLCF: Directional local contrast filtering

Lens artifacts saturate a fundus image's luminance, resulting in regions of high contrast with respect to the local background. We make use of this visual saliency to detect and remove these artifacts. Also, as we illustrate in Figure A.3, our filtering approach is also able to remove certain kinds of non-vessel retinal features, such as

exudates (Osareh et al., 2003).

We model saliency based on Weber's measure of contrast (Peli, 1990):

$$c_o = \frac{m_o - m_b}{m_b}, \tag{A.1}$$

where m_o is the median intensity of the object and m_b is the median intensity of the background. We use the median value, as opposed to the mean, due to its greater robustness to outliers. The Weber contrast measure is defined for grayscale images. We apply it to RGB fundus images by considering only the green channel, as is standard practice for retinal images (Soares et al., 2006) because of its stronger vessel contrast. We define a local Weber contrast measure for each individual pixel $\mathbf{p}$ by determining m_b from a small, rectangular neighborhood around $\mathbf{p}$:

$$c_{\mathbf{p}} = \frac{I^g(\mathbf{p}) - m_b^g(\mathbf{p})}{m_b^g(\mathbf{p})}, \tag{A.2}$$

where the g superscript indicates the green channel. The exact value of $m_b^g(\mathbf{p})$ depends on the size of the neighborhood window. In our experiments, results were robust to variations in window size, as long as the window is larger than the targeted artifacts.

The sign of $c_{\mathbf{p}}$ is different for bright ($c_{\mathbf{p}} > 0$) and dark ($c_{\mathbf{p}} < 0$) contrast. Regions of dark contrast include the vessels. Therefore, we do not modify these pixels. The only anatomical region in retinal images with bright contrast is the optic nerve head, which is not part of the vascular network. We therefore identify pixels with bright contrast that exceed a threshold value t_c and replace them with the corresponding median value:

$$c_{\mathbf{p}} > t_c \Rightarrow I(\mathbf{p}) \leftarrow m_b(\mathbf{p}) \; . \tag{A.3}$$

Figure A.2 illustrates the effects of these operations. The threshold value t_c was determined empirically in our experiments. In multi-frame fusion (Estrada et al.,

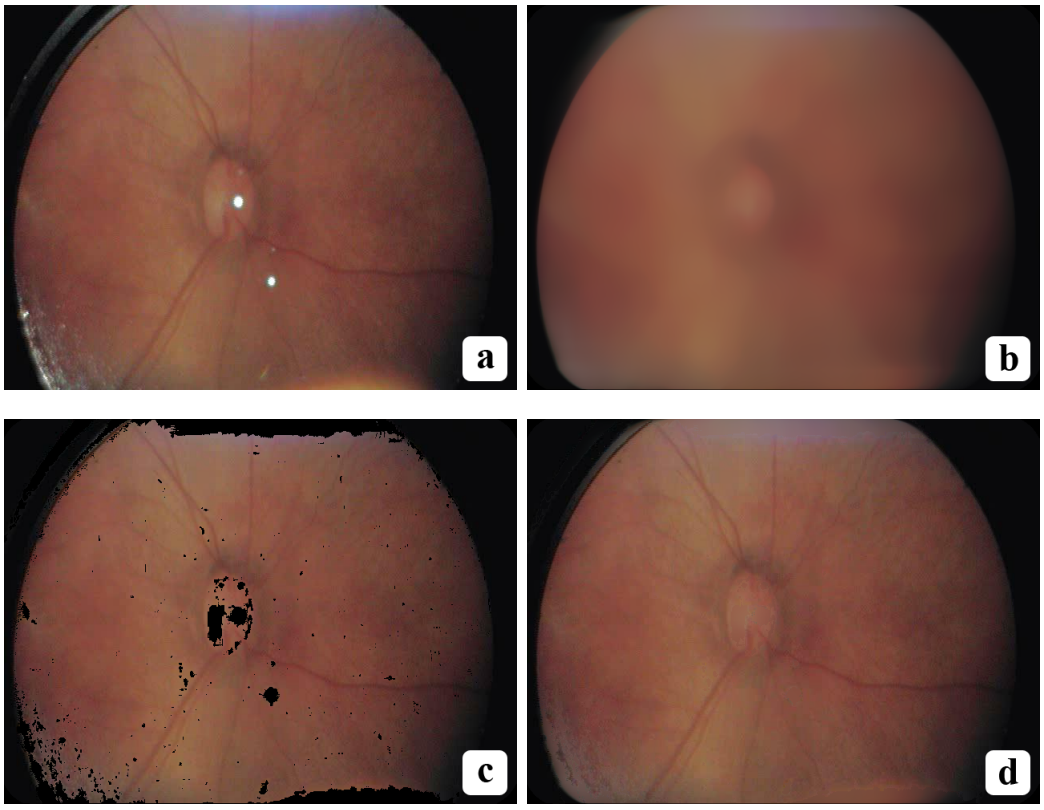


FIGURE A.2: **The steps of directional local contrast filtering: (a)** The original image. **(b)** The local median for a 50 × 50 pixel filtering window. **(c)** Pixels that far exceed the local median brightness are marked as invalid (black in the image). **(d)** Invalid pixels are replaced with the local median values. This removes white spots and speckles.

2011), bright contrast pixels can also be replaced with the corresponding pixel values from overlapping frames instead of the local median value.

A.1.2 HSV classification

As noted above, all regions of the image that correspond to our tree of interest, the vascular network, fall within the retina. We thus first isolate the retina before isolating the tree itself. We classify pixels in the Hue-Saturation-Value (HSV) color space (Gonzalez and Woods, 2002) to assign a retinal or non-retinal label to every pixel. Classification in HSV space is more robust to highlights, shadows, and texture variations than in other color spaces (Cheng et al., 2001). We use color for pixel

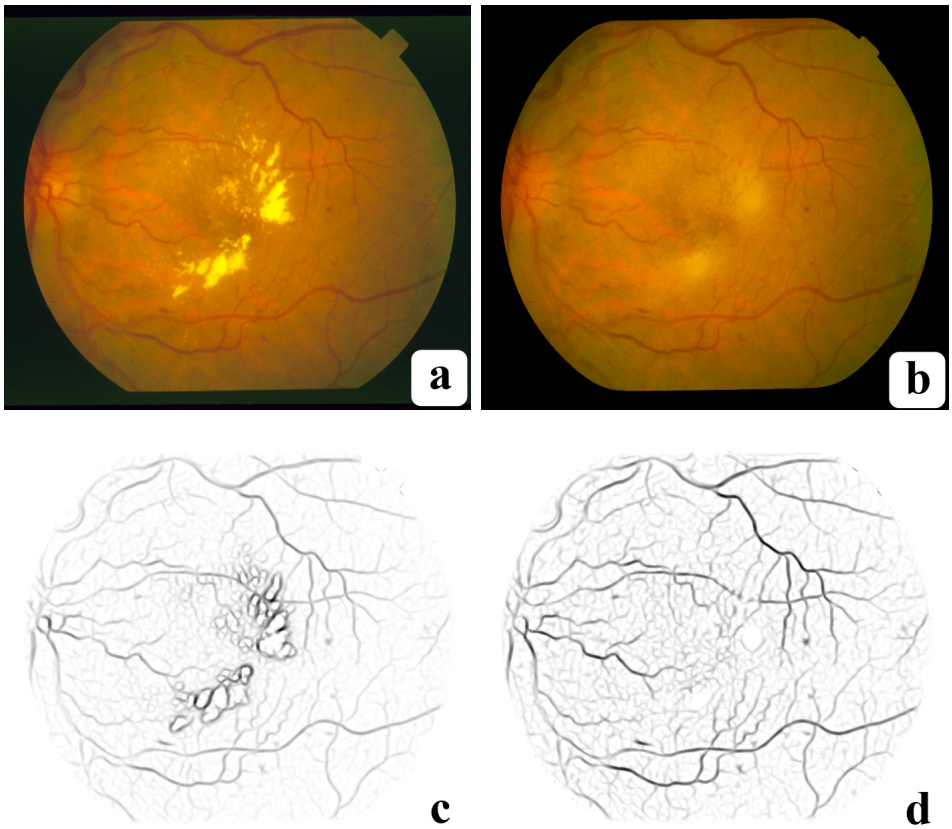


FIGURE A.3: **DLCF exudate removal: (a)** An image from the STARE dataset (Hoover et al., 2002). **(b)** The image after DLCF. **(c)** Matched filtering (Chaudhuri et al., 1989) applied to (a). **(d)** Matched filtering applied to (b). The non-vascular filter responses around the exudates have been eliminated in (d) without affecting the true vessel responses.

classification because we empirically determined that retinal pixels for a given eye exhibit a narrow and consistent color distribution. Furthermore, the color distribution of any image can be determined reliably and efficiently.

In HSV classification, we specify a closed decision region R_{HSV} in HSV space. Vectors within R_{HSV} are classified as retinal pixels, while those outside R_{HSV} are labeled non-retinal. We specify R_{HSV} as the Cartesian product of three intervals, R_H, R_S, and R_V. We eschew more complex boundaries for computational efficiency.

Let $\mathcal{I}$ be the set of pixels in I. Then, we construct the set $\mathcal{I}_{\text{HSV}}$ of retinal pixels as those whose HSV values lie within R_{HSV}. An image's HSV score is given by the

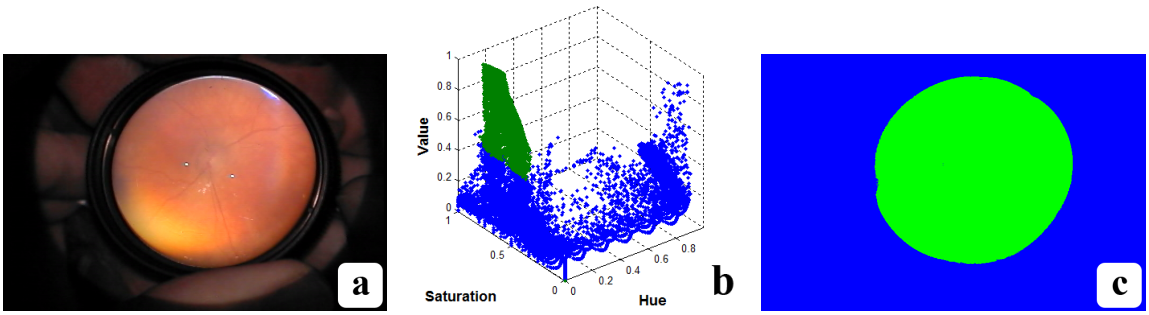


FIGURE A.4: **HSV color distribution: (a)** A sample retinal image. **(b)** The scatter plot of the HSV color values of the sample frame. Retinal pixels (green), exhibit a narrow color distribution in HSV space relative to the rest of the image (blue). While the retinal pixels constitute 30% of the image, they are more tightly clustered than the non-retinal pixels. **(c)** Color-coded frame: retinal pixels are shown in green and non-retinal pixels in blue.

proportion of retinal pixels in it:

$$h(I) = \frac{|\mathcal{I}_{\text{HSV}}|}{|\mathcal{I}|}. \tag{A.4}$$

where the bars denote set cardinality. Fig. A.4 shows a sample color distribution. After artifact removal, we mask any pixels which fall outside the HSV boundary R to remove the remaining spurious objects such as the physician's hands, the condensing lens' disk, and other surrounding objects, as shown in Fig. A.5.

A.2 Branch enhancement

Given an image I, we wish to determine a likelihood map $I_{\mathcal{L}}$, such that $I_{\mathcal{L}}(\mathbf{p})$ is the likelihood that pixel $\mathbf{p}$ is part of a tree. The simplest likelihood map consists of the original pixel values $I_{\mathcal{L}} = I$. However, for many images, this likelihood is quite poor. In many images, particularly vascular ones, there is often very little contrast between the tree's branches and the surrounding background, even after removing extraneous objects and artifacts. Thus, it can be quite difficult to reliably detect the tree pixels directly.

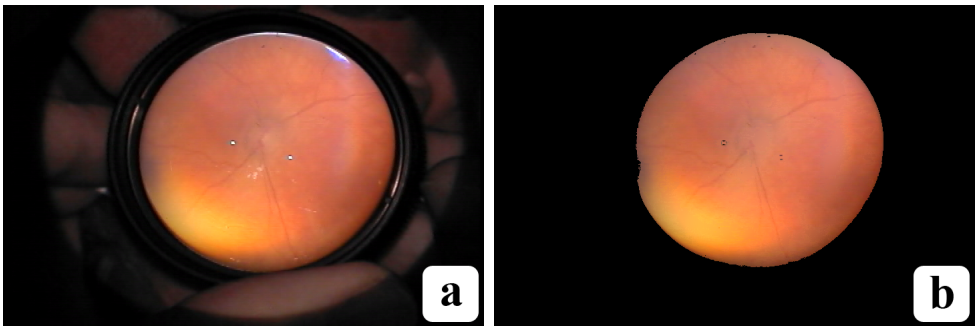


FIGURE A.5: **HSV masking:** Pixels in **(a)** that fall outside the HSV boundary R are flagged and discarded from further processing. Non-retinal pixels are shown as black in **(b)**.

In this section, we propose a pixel likelihood map which consists of each pixel's response to two sets of filter banks. Via a two-step algorithm, we maximize the contrast between tree and non-tree pixels. Our method is particularly well-suited for eye vessel images, but can be readily applied to various other types of tree images, such as plant roots.

General contrast enhancement methods (Starck et al., 2003; Acharya and Ray, 2005) are insufficient for distinguishing between tree and non-tree pixels because we wish to selectively identify only the tree's branches and not other parts of the image. Thus, we enhance the detectability of the tree's branches through a multi-scale approach using Laplacian-of-Gaussian (LoG) filters and Gabor wavelets (Movellan, 2002). In the first step, filtering the image with a multiscale LoG kernel produces a high-contrast tree map image. In the second step, filtering the resulting map with a bank of Gabor filters solidifies branch connectivity. As noted above, we interpret each pixel's response to the LoG and Gabor filter banks as the likelihood that that pixel is part of the tree.

Filtering kernels are widely used to enhance retinal vessels (Chaudhuri et al., 1989; Soares et al., 2006; Li et al., 2006; Wilson et al., 2008; Kirbas and Quek, 2004) due to the vessels' marked elongation and their narrow intensity distribution relative to

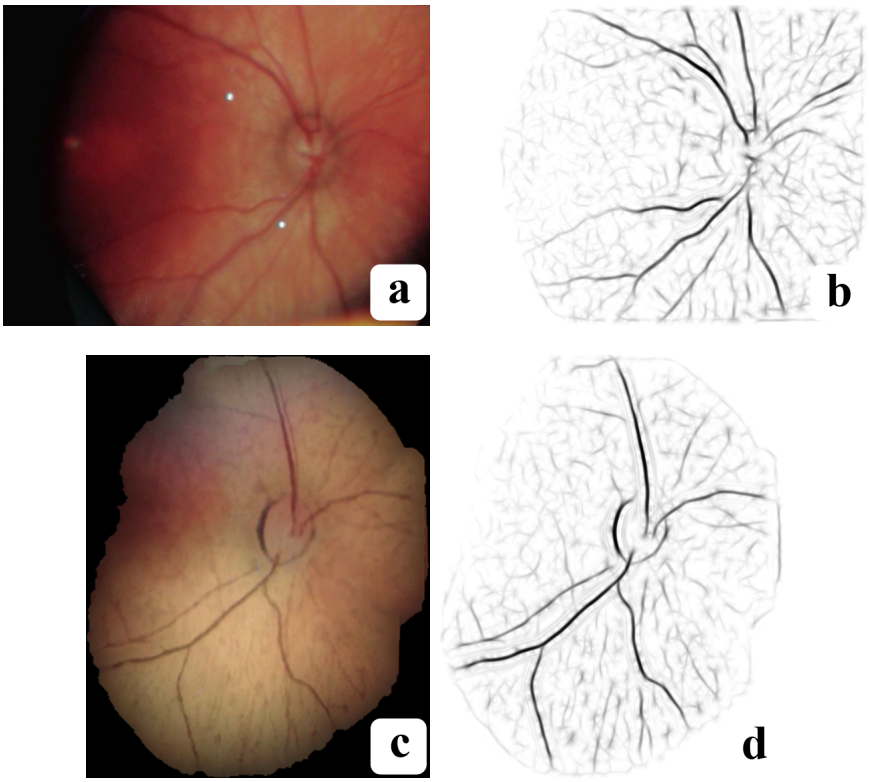


FIGURE A.6: **LoG-Gabor filtering:** **(a)** A sample fundus image. **(b)** Image after LoG-Gabor filtering. **(c)** A sample retinal mosaic from Estrada et al. (2011). **(d)** Mosaic after LoG-Gabor filtering. The isotropic LoG filtering enhances vessel contrast, while the anisotropic Gabor wavelets selectively enhance elongated structures.

the surrounding tissue. As noted above, many other kinds of trees, such as lightning and plant roots, also have elongated branches and thus can also be enhanced in this way.

A.2.1 LoG filter bank

Conceptually, the LoG filter is obtained by first low-pass filtering the original image with a zero mean Gaussian kernel:

$$\mathcal{N}_\sigma = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(\mathbf{p}-\mu)^2/2\sigma^2}, \tag{A.5}$$

where a single scale value σ controls the amount of smoothing. In the next step, the contrast-sensitive Laplace operator

$$\nabla^2 = \frac{\partial^2}{\partial \mathbf{p}_x^2} + \frac{\partial^2}{\partial \mathbf{p}_y^2} \tag{A.6}$$

is applied to the resulting image:

$$L_\sigma(\mathbf{p}) = \nabla^2 (I^g(\mathbf{p}) * \mathcal{N}_\sigma(\mathbf{p})). \tag{A.7}$$

I^g is the green channel of the RGB image I.

Since convolution and Laplacian are linear operations, a more practical scheme applies the Laplacian to the Gaussian kernel, and the resulting operator is then convolved with I^g in a single pass:

$$L_\sigma(\mathbf{p}) = (\nabla^2 \mathcal{N}_\sigma(\mathbf{p})) * I^g(\mathbf{p}). \tag{A.8}$$

Since the Laplacian is the divergence of the gradient, a strong, positive filter response at a given pixel indicates the presence of a dark pixel surrounded by lighter ones, a pattern which is often produced by a tree's branches. The parameter σ determines the scale at which the gradient is detected, smaller for thinner branches. To account for scale, we utilize a bank of LoG filters between a minimum and maximum σ and a fixed step size Δ. We convolve the original image with each filter in the bank and retain the bank's maximal response at every pixel:

$$L(\mathbf{p}) = \max_\sigma L_\sigma(\mathbf{p}). \tag{A.9}$$

For visualization, the complement image

$$\hat{L}(\mathbf{p}) = \max_{\mathbf{q}}(L(\mathbf{q})) - L(\mathbf{p}) \tag{A.10}$$

is often more intuitive than L, because many types of trees, such as vessels an and plant roots appear dark in $\hat{L}$ but show up as light (strong response) in L. The result of convolving a frame with a multiscale LoG filter bank is shown (as a complement image) in Fig. A.6 (b).

A.2.2 Gabor wavelet bank

As Fig. A.6 illustrates, the branches in the LoG filtered image have higher contrast relative to the original image. The LoG filtered image also eliminates low-frequency illumination differences between the various parts of the image. However, in low SNR regions, tree branches tend to produce weaker filter responses and are thus broken up in segments. To address this problem, we make use of Gabor wavelets, defined by multiplying a complex sinusoid by a Gaussian kernel (Movellan, 2002):

$$W(\mathbf{p}) = sn(\mathbf{p})\mathcal{N}_\Sigma(\mathbf{p}), \tag{A.11}$$

where $sn(\mathbf{p})$ is the sinusoidal component and $\mathcal{N}(\mathbf{p})$ is an anisotropic, scaled, and rotated Gaussian function:

$$s(\mathbf{p}) = e^{i(2\pi(ux+vy)+\alpha)}, \qquad \mathcal{N}_\Sigma(\mathbf{p}) = \beta e^{-\pi(x_\theta^2/\sigma_a^2 + y_\theta^2/\sigma_b^2)}. \tag{A.12}$$

Here, u and v determine the spatial frequency of the sinusoid, while α indicates its phase. For the Gaussian function, β is an overall scaling factor, while σ_a^2 and σ_b^2 each control the variance along a single axis. θ indicates the angle of rotation:

$$x_\theta = x\cos\theta + y\sin\theta, \qquad y_\theta = y\cos\theta - x\sin\theta\ . \tag{A.13}$$

Different input parameters produce Gabor wavelets of different scales and orientations. We build a Gabor wavelet bank by systematically varying each of the parameters over finite intervals. For σ_a, σ_b, u and v, these intervals encompass the range of expected branch widths. We convolve each wavelet in the bank with the input image and keep the maximum response at each pixel:

$$F(\mathbf{p}) = \max_{u,v,\alpha,a,b,\theta,\beta}(L(\mathbf{p}) * W(\mathbf{p})). \tag{A.14}$$

Unlike LoG kernels, our Gabor wavelets are anisotropic, and this allows them to selectively enhance elongated regions. As Fig. A.6 (d) shows, branches not only display high contrast, but better connectivity. As with a LoG filtered image, the

Log-Gabor filtered image F retains the maximum filter response at any of the scales and orientations, and $\hat{F}$ denotes the complement image of F. Fig. A.6 (c) shows a $\hat{F}$ image.

Thus, for the remainder of this work, we define the likelihood of each pixel to be part of the tree in terms of the LoG-Gabor filtered image:

$$I_{\mathcal{L}} = F.$$

In the next chapter, we will explore how to convert this likelihood map into a binary segmentation via dynamic programming.

Appendix B

Image Segmentation

Segmentation is arguably the most studied problem concerning images of trees. Here, given a source image I, we wish to determine a binary mask I_B, such that:

$$I_B(\mathbf{p}) = \begin{cases} 1 & \text{if } \mathbf{p} \text{ is part of the tree} \\ 0 & \text{if } \mathbf{p} \text{ is not} \end{cases}$$

In this work, we wish to isolate those pixels that belong to the imaged tree. As noted in Appendix A, the exact procedure required to isolate a specific tree is dependent on both the tree in question, as well as the imaging device and conditions used to acquire the image. Nevertheless, in this chapter we will detail a tree segmentation method that has proven effective on retinal images (Estrada et al., 2012).

As detailed in Chapter 2, retinal segmentation methods fall into two main categories: tracking (path-based) and filtering (region-based). Our hybrid approach extends the path-based methodology into a region-based segmentation scheme for detecting retinal vessels. Our complete approach works in two stages, as illustrated in Fig. A.1. As we showed in Appendix A, the first stage pre-processes the input image to remove both lens and motion artifacts, and to construct a high-contrast

likelihood map.

As we will now show, the second stage builds a forest of tree regions through a sequence of exploration waves on the vessel map: the most tree-like pixel $\mathbf{s}_0$ in the image is used as the starting point for an exploration wave that searches for the best tree region in the image around $\mathbf{s}_0$ by means of the single-source, multi-destination version of Dijkstra's shortest path algorithm (Dijkstra, 1959). This exploration returns an entire *region* for part of the vessel system, not just a single path; that is, it handles branching naturally and efficiently, and preserves branch thickness. When this exploration ends, a new exploration begins at the best remaining starting point $\mathbf{s}_1$ in the unexplored part of the image, which yields a new tree region. Our method stops constructing new regions when the best unexplored starting point is no longer likely to be part of the tree.

Unlike existing single-source, single-destination vessel analysis methods (Benmansour and Cohen, 2011; Li and Yezzi, 2007; Pechaud et al., 2009; Wink et al., 2004), our single-source, multiple-destinations approach automatically explores the complete vasculature in a retinal image, and requires no user intervention whatsoever.

B.1 Method overview

We represent each image I as a graph $G_I = (V_I, E_I)$, where each vertex corresponds to a pixel and edges connect neighboring pixels. In this formulation, the ordered pair of vertex and edge sets are represented by V_I and E_I, respectively. Path-based methods for vessel extraction define the *cost* of traversing the edge that connects any two neighboring pixels in the image in such a way that edges between tree pixels are more likely to have lower cost. They then looks for paths that traverse the image from neighbor to neighbor and have minimum aggregate cost, and are thereby likely to follow branches. If the cost aggregation rule is associative, minimum-cost paths

can be found efficiently (Bellman, 2003).

We depart from previous work within this framework in two major ways. First, we find tree *regions*, rather than simply tree paths. In other words, we preserve branch thickness, rather than merely finding the skeleton, or centerline, of each branch. Second, we employ a sequence of searches for tree regions that start at *source points* $\mathbf{s}_0$, $\mathbf{s}_1$, ... automatically selected in decreasing order of their likelihood to be part of a vessel, as detailed in Subsection B.5. This novelty eliminates the need for a user to select starting points by hand.

Thus, we use the single-source, multiple-destination version of Dijkstra's shortest path algorithm (Dijkstra, 1959) rather than the single-source, single-destination version used in prior work. In other words, rather than *connecting* a start point with a destination point, our method *explores* the image outward from an (automatically selected) source point. This exploratory strategy has two advantages: it eliminates the need for selecting a destination point manually, and it finds tree-like image regions, thereby accounting for branching naturally and efficiently.

The computational cost of this important change of perspective is trivial, as the only difference between the single-destination and multi-destination algorithms is when they stop: the single-destination algorithm stops when it reaches the designated vertex, while the multi-destination algorithm stops when a target threshold on the path cost has been reached. Both versions of Dijkstra's algorithm have the same computational complexity of $O(|E| + |V| \log |V|)$, where $|\cdot|$ indicates the cardinality or size of a set. This complexity is achievable with a heap-based priority queue implementation (Cormen et al., 2001).

B.2 Edges and edge costs

Prior to obtaining the graph G_I, we first remove the image's artifacts by using directional local contrast filtering (DLCF) as defined in Appendix A. Figure A.3

illustrates the effect of this image enhancement step.

We then define two features that determine the edge costs at each pixel $\mathbf{p}$: the green channel intensity $I^g(\mathbf{p})$ and the inverted response $F(\mathbf{p})$ to a Laplacian-of-Gaussian filter followed by a Gabor filter bank, or LoG-Gabor filtering, as detailed in Appendix A. The vessel map F maximizes the discriminability of vessels, as illustrated in Fig. A.6.

To apply Dijkstra's algorithm to I, we define a weighted lattice graph on the set $V_I = \{\mathbf{p}\}$ of all pixel locations in the image. There is an edge $e = (\mathbf{v}, \mathbf{v}')$ in the edge set E_I for this directed graph $G_I = (V_I, E_I)$ for any ordered pair of 8-neighbors, that is, whenever

$$\max(|x - x'|, |y - y'|) = 1 \; . \tag{B.1}$$

A non-negative cost is defined on each edge, with the intent that edges inside and along vessels cost less than edges that have one or both endpoints outside any vessel. Specifically, we define the cost of edge e as the following convex linear combination:

$$c(e) = \sum_{m=1}^{4} w_m e^{\alpha z_m(e)} \qquad \text{where} \qquad \sum_{m=1}^{4} w_m = 1 \tag{B.2}$$

and $z_m(e)$ indicates the m-th element in the following four-dimensional feature vector:

$$\mathbf{z}(e) = \mathbf{z}(\mathbf{v}, \mathbf{v}') = \Big[I^g(\mathbf{v}'),\, |I^g(\mathbf{v}) - I^g(\mathbf{v}')|,\, F(\mathbf{v}'), |F(\mathbf{v}) - F(\mathbf{v}')| \Big] \; . \tag{B.3}$$

Therefore, a low-cost edge is an edge whose destination point $\mathbf{v}'$ is dark ($I^g(\mathbf{v}') << 1$) and has a low inverted LoG-Gabor response ($F(\mathbf{v}') << 1$), and such that the two edge endpoints are similar in both brightness ($|I^g(\mathbf{v}) - I^g(\mathbf{v}')| << 1$) and LoG-Gabor response ($|F(\mathbf{v}) - F(\mathbf{v}')|$).

The exponential in Eq. B.2 provides a non-linear scaling of the edge's features that emphasizes the divide between branch and non-branch feature values, and the scalar α controls the growth rate of the exponential term.

Algorithm 4: Exploratory Dijkstra vessel segmentation: starting from a single pixel, the algorithm progressively explores the rest of the image such that every unvisited pixel has a higher minimum path cost than every visited pixel. The algorithm keeps adding pixels until a cost boundary is reached.

Input: Graph G, source vertex $\mathbf{s}$, threshold τ.
Output: Dijkstra region R.

```
Q = initialize_priority_queue();
push(Q,s,0);
while not_empty(Q) do
    [v_c, γ̃_{0,c}] = pop(Q);
    if not_visited(v_c) then
        set_visited(G,v_c);
        V_c = neighbors(G,v_c);
        foreach v ∈ V_c do
            h = c(γ̃(s, v_c)) + c(v_c, v);
            if h < τ then
                push(Q,v,h);
R = visited(G);
```

B.3 Path costs

A path γ between any two vertices $\mathbf{v}$, $\mathbf{v}'$ in V_I is composed of a sequence of neighboring lattice locations:

$$\gamma(\mathbf{v}, \mathbf{v}') = (\mathbf{v} = \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k = \mathbf{v}'), \tag{B.4}$$

subject to the constraint that $(\mathbf{v}_i, \mathbf{v}_{i+1}) \in E_I$, for $i \in [1, k-1]$. In short, γ is a curve discretized as a sequence of neighboring pixels. The cost of γ is defined as the sum of the costs of its edges:

$$c(\gamma) = \sum_{i=1}^{k-1} c(\mathbf{v}_i, \mathbf{v}_{i+1}) \,. \tag{B.5}$$

The associative nature of this definition allows splitting a path's total cost into disjoint sub-path costs at any point along γ:

$$c(\gamma(\mathbf{v}, \mathbf{v}')) = c(\gamma(\mathbf{v}, \mathbf{v}_i)) + c(\gamma(\mathbf{v}_i, \mathbf{v}')) \qquad \text{for any} \qquad i \in [2, k-1] \,, \tag{B.6}$$

with which we can efficiently determine the minimum cost path between any two vertices $\mathbf{v}$ and $\mathbf{v}'$. That is, we use Dijkstra's algorithm to compute:

$$\tilde{\gamma}(\mathbf{v}, \mathbf{v}') = \underset{\gamma \in \Gamma(\mathbf{v}, \mathbf{v}')}{\operatorname{argmin}} c(\gamma), \tag{B.7}$$

where $\Gamma(\mathbf{v}, \mathbf{v}')$ is the set of all possible paths between the two vertices.

B.4 Exploratory Dijkstra segmentation

Dijkstra' minimum cost algorithm solves (B.7) for any graph with non-negative edge costs (Dijkstra, 1959). More generally, it finds a minimum cost path $\tilde{\gamma}(\mathbf{s}, \mathbf{v})$ between a single source vertex $\mathbf{s}$ and (potentially) every other vertex $\mathbf{v}$ in the graph.

As discussed earlier, instead of simply connecting user-defined points, we employ an *exploratory* strategy by using the single-source, many-destinations version of Dijkstra's method. Starting from a single position $\mathbf{r}$ on a major branch, this strategy enables us to segment this major branch and all the less prominent branches that branch out of it, without any need for setting any destination point. Instead, we set an exploration threshold τ on the cost of any path, and find all the minimum-cost paths $\tilde{\gamma}$ from $\mathbf{r}$ in G such that $c(\tilde{\gamma}) < \tau$. Algorithm 4 outlines our exploratory Dijkstra tree segmentation method.

With the lattice edge costs defined in (B.2), the exploratory Dijkstra algorithm will preferentially visit tree pixels before exploring non-tree ones, since the cost to reach the latter is generally much higher. When it stops, it will have visited the *Dijkstra region*:

$$R_\tau(\mathbf{r}) = \{\mathbf{v} \,|\, \tilde{\gamma}(\mathbf{r}, \mathbf{v}) \leqslant \tau\} \,. \tag{B.8}$$

The segmentation's accuracy is thus dependent on the value of τ. However, our choice of τ is made less sensitive by the exponential in (B.2), which increases the separation between the tree and non-tree pixel classes. This lower sensitivity reduces both

the problem of "leakage", in which a segmentation goes beyond the correct branch boundary and the problem of stopping too soon.

Algorithm 5: Dijkstra forest tree segmentation: The algorithm adds disjoint Dijkstra regions until the minimum inverted LoG-Gabor response at the source pixel exceeds ψ. The operation $V_I \setminus \mathbf{R}$ represents $\{v \in V_I \mid v \notin \mathbf{R}\}$.

Input: Graph G_I over image domain V_I, inverted LoG-Gabor responses F, exploratory threshold τ, filtering threshold ψ.
Output: Dijkstra forest $\mathbf{R}$.
$\mathbf{R} = \varnothing$;
while $\mathbf{r} < \psi$ **do**
  $\mathbf{r} = \texttt{argmin}_{V_I}(F)$;
  $R = \texttt{exploratory\_dijkstra}(G_I, \mathbf{r}, \tau)$;
  $\mathbf{R} = \mathbf{R} \cup R$;
  $V_I = V_I \setminus \mathbf{R}$;

B.5 Dijkstra forest

The exploratory Dijkstra method outlined in Subsection B.4 efficiently segments a Dijkstra region $R_\tau(\mathbf{r})$ given a single source vertex $\mathbf{r}$. As Fig. A.6 (d) exemplifies, however, the branches in a tree can extend from more than one primary branch. Furthermore, low image quality and blur can obscure large sections of the tree, and break it up into several disconnected regions. Therefore, in order to segment all visible branches better, we extend the single source method to multiple sources.

To this end, we first generate the initial region $R_0 = R_\tau(\mathbf{r}_0)$ from a first source point $\mathbf{r}_0$ as described above. We then select a new source vertex $\mathbf{r}_1$ from those vertices in V_I that are not part of R_0, and generate a new region R_1 from it, such that $R_0 \cap R_1 = \varnothing$. By repeating, we thus form a *Dijkstra forest*:

$$\mathbf{R} = \{R_0, R_1, \cdots, R_K\}, \qquad \text{where} \qquad F(\mathbf{r}_0) \leqslant \ldots \leqslant F(\mathbf{r}_K) \leqslant \psi. \tag{B.9}$$

Here, ψ is a threshold on the highest allowable inverted LoG-Gabor response. We stop adding new regions to the forest when the highest response outside $\mathbf{R}$ is higher than ψ. Algorithm 5 outlines the complete Dijkstra forest computation. In

Figure B.1, we show the results of applying our method compared to other state-of-the-art vessel segmentation methods, as explored further in (Estrada et al., 2012).

In the previous appendix, we detailed how to remove a number of image artifacts and enhance the branches of a tree. Here, we presented a novel dynamic programming methodology for segmenting the pixels that correspond to the tree. In the following appendix, we will show how to estimate a planar graph given a segmented binary image.

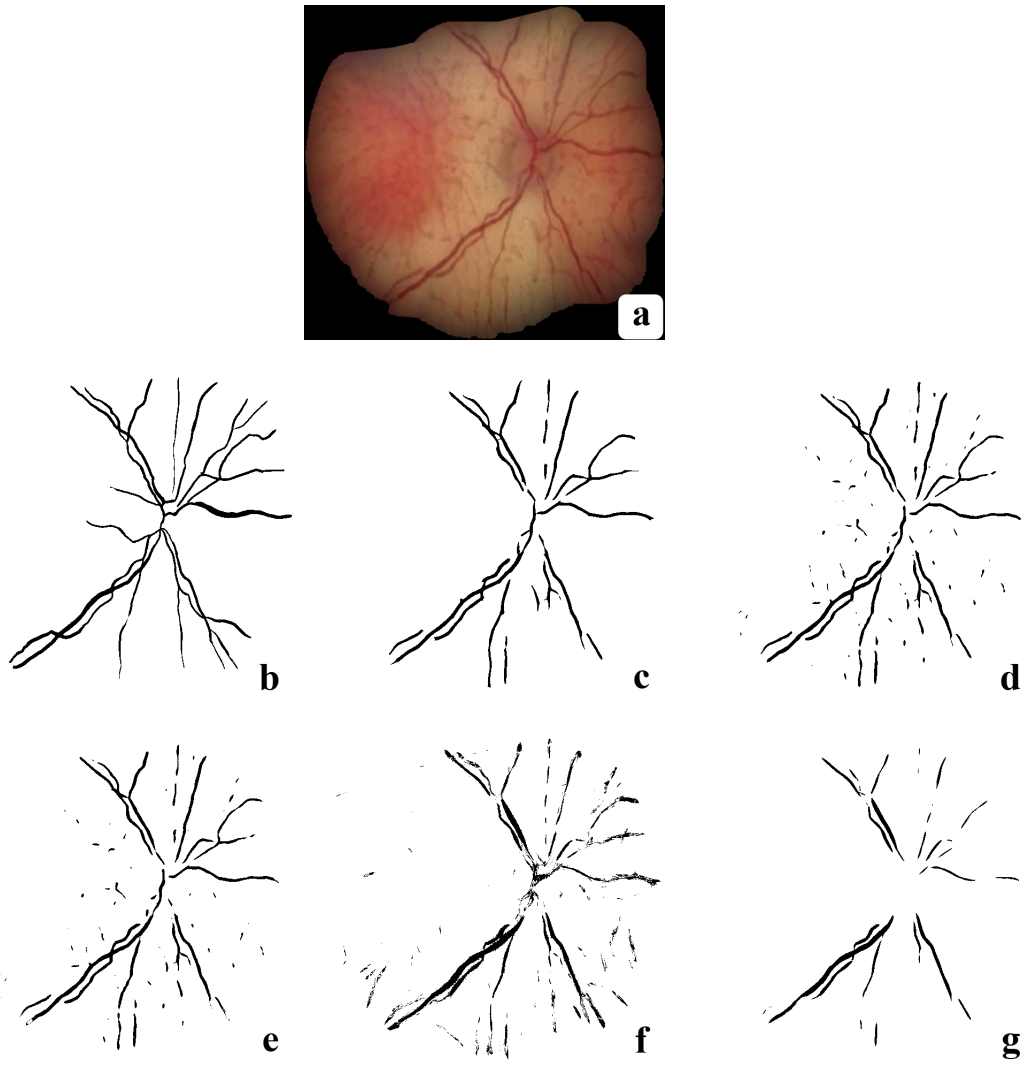


FIGURE B.1: **Vessel segmentation on a mosaic image:** Our Dijkstra segmentation method outperformed other state-of-the-art vessel segmentation approaches on a dataset consisting of video indirect ophthalmoscopy retinal images (Estrada et al., 2012). **(a)** A sample image **(b)** Manual segmentation **(c)** Dijkstra forest **(d)** Matched filters **(e)** Local entropy **(f)** GMM classifier **(g)** KNN classifier

Appendix C

Graph Estimation

C.1 Graph estimation

In Chapter 3, we formulated the general problem of determining a tree given a planar graph and in subsequent chapters, we presented a methodology for determining the most likely tree given a particular graph. In the previous chapters, we assumed the input graph $G = (V, E)$ as given. In practice, however, determining G given an image is itself a challenging problem. In this chapter, we detail a tracking methodology for estimating G given a segmented image mask, such as one obtained by the Dijkstra segmentation method defined in Appendix B.

Our approach is based on segmenting the tree into a set of *tracks*. Each track is a sequence of finite support *segments*. Our track segmentation technique uses a combination of shape, orientation, and radiometric features to directly encode the tree's branches. It allows us to estimate the location of end- and branch-points, as well as branch-crossings. Given a set of tracks over an image, we construct a graph based on where each segment is located and which other segments it overlaps with. We will first describe our track segmentation in the following sections and then detail

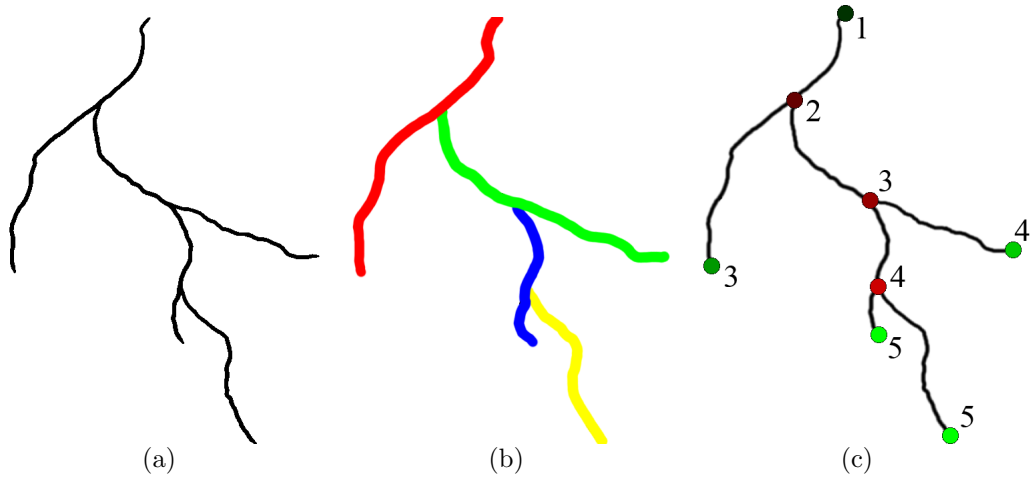


FIGURE C.1: **Track segmentation:** A tree's topology is encapsulated in the hierarchical relations between its key-points (end-, and branch-points). **(a)** A simple tree. **(b)** A set of tracks over the tree. Each track is indicated by a different color. **(c)** The automatically detected end-points (in green) and branch-points (in red). The darker the circle, the higher up in the hierarchy. The number indicates the depth relative to the root. Branch-points at located where two or more tracks meet.

how to determine the graph from the tracks in Section C.4.

For most images, only a subset of pixels will be part of a meaningful tree. As noted in Appendix B, in a binary segmentation, we assign each pixel one of two labels: tree or not-tree. In the following discussion, we restrict our analysis to the pixels that fall within this binary mask.

A tree's *topology* consists of the hierarchical relations between its end- and branch-points, which define how the tree's branches are connected to each other. In contrast, a tree's *geometry* describes the spatial characteristics of the tree, including the location, thickness, orientation and tortuosity of its branches. In most physical trees, branch-points are location that indicate the intersection of two or more branches, not distinct physical parts of the tree. In other words, it is easier to detect branches than branch-points. Therefore, we first estimate the tree's branches using a set of tracks and then estimate the locations of branch-points by observing where the tracks intersect.

C.2 Track definition

To estimate a tree's topology, we construct a set of tracks that span the tree's branches. The union of the estimated tracks corresponds to the initial binary segmentation, while the points of contact between the tracks implicitly define the location of the tree's branch-points and branch-crossings. We define a track as a sequence of connected two-dimensional *track segments*:

$$\mathbf{S} \equiv \{S_1, S_2, \cdots, S_T\}, \tag{C.1}$$

where S_t is the t-th segment. A track segment is a finite support function determined by a three-element tuple $[\mathbf{c}, \rho_{\mathbf{c}}, \delta\mathbf{c}]$, where $\mathbf{c}$ is the spatial location of the segment's center, $\rho_{\mathbf{c}}$ is the segment's radius, and $\delta\mathbf{c}$ indicates the direction from which the segment was reached. The radius can be constant or adaptively dependent on local image statistics. Unlike a skeleton, successive center pixels will not be contiguous. Given a tuple, the associated segment is an image mask that selects only pixels that lie within $\rho_{\mathbf{c}}$ of $\mathbf{c}$ and are part of the binary mask I_B:

$$S(\mathbf{p}) = \begin{cases} I(\mathbf{p}) & \text{if } \|\mathbf{p}-\mathbf{c}\|_2 \leqslant \rho_{\mathbf{c}} \quad \text{and} \quad I_B(\mathbf{p}) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

In our formulation, we impose three constraints on the possible sequence of segments in a track. Let $\mathcal{S}_t$ be the set of pixels with non-zero values for S_t. Then,

$$\mathcal{S}_{t+1} \cap \mathcal{S}_t \neq \varnothing \tag{C.2}$$

$$\mathcal{S}_{t+1} \cap \mathcal{S}_{t-1} = \varnothing \tag{C.3}$$

$$\mathcal{S}_{t+1} \neq \mathcal{S}_t. \tag{C.4}$$

The first constraint implies that the union between the pixels of successive segments has to be non-empty. Thus, a track is a connected region in I. The second and third constraints prohibit a track from oscillating between adjacent regions.

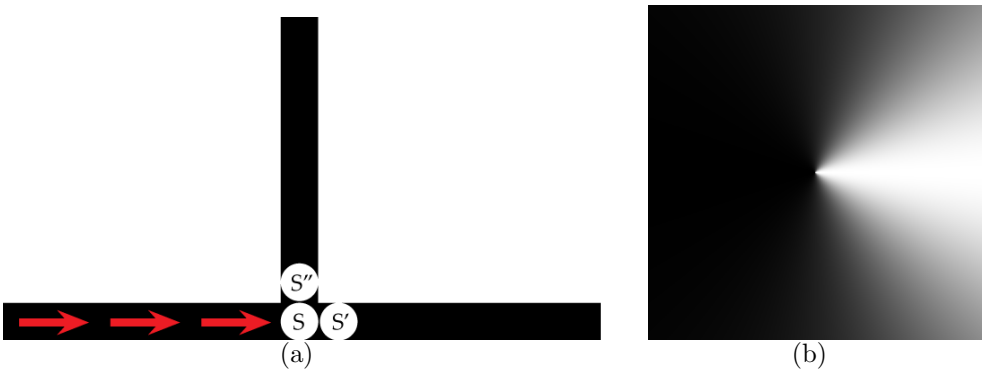


FIGURE C.2: **von Mises directional prior: (a)** A physical object moving along the black tree in the direction indicated by the red arrows will continue on to segment S', even though S' and S'' have the same local appearance. Our kernels model this motion with momentum along branches via a von Mises directional prior. **(b)** Von Mises kernel. White indicates a higher probability. Segments lying in the current direction of motion are preferred to more orthogonal segments.

As with other forms of image tracking, successive segments, to a certain extent, should be similar to each other. Furthermore, since branches are generally tubular structures, we require that the directions $\delta\mathbf{c}_t$ and $\delta\mathbf{c}_{t+1}$ from which successive segments were reached to be similar, as defined below.

C.2.1 Segment orientation

Successive track segments should be similar to each other, which naturally implies they should share similar pixel values. In a tree, however, the likelihood that two nearby regions belong to the same branch is not defined solely by their appearance, but also by their spatial relationship. Figure C.2 illustrates this principle. Locally, S' and S'' have identical pixel values. However, if we view S as a ball rolling along the black track in the direction indicated by the red arrows, S' is a more intuitive choice than S''. Similarly, in our formulation, we construct tracks by moving a kernel with momentum along a tree's branches.

To formalize the intuition that a track should favor straight paths, we make use of a von Mises distribution that weighs potential segments by their angle with respect to

the direction along which the current segment was reached. As defined in Chapter 8, the von Mises distribution is a probability distribution defined on the unit circle:

$$\mathrm{VM}(\mathbf{v}; \boldsymbol{\mu}, \kappa) = \frac{e^{(\kappa \boldsymbol{\mu}^\top \mathbf{v})}}{2\pi J_0(\kappa)}. \tag{C.5}$$

Here, we extend the above equation to a kernel on the plane through polar coordinates: let $[r, \theta]$ be the polar coordinates of the vector $\mathbf{v}$, centered at $\mathbf{c}$. Then, all vectors with the same θ are assigned the same probability, regardless of r. Figure C.2 (b) illustrates the central part of a two-dimensional von Mises kernel.

C.3 Track estimation

We construct each track by first identifying an initial seed location, and then iteratively minimizing the orientation-weighted sum-of-squared-differences (SSD) between the current segment and a local neighborhood of potential next segments, while obeying the growth constraints of (C.2), as defined below. A track is grown in two directions: one towards the root and one towards an end-point.

C.3.1 Track seed selection

In Appendix A, we detailed how to enhance a tree's branches using LoG-Gabor filtering. The response of each pixel provides a local estimate of how likely that pixel is to be part of the tree. We make use of these responses to estimate the initial location of each track.

Let F be the image of LoG-Gabor responses. Then, To find a new seed location $\mathbf{c}_k$, we select a maximum LoG-Gabor response from the pixels that have not yet been visited by any track:

$$\mathbf{c}_k = \underset{I_B \setminus \mathbf{s}_{1:k}}{\operatorname{argmax}} F(\mathbf{p}) \ , \tag{C.6}$$

where $I_{B \setminus \mathbf{s}_{1:k}}$ are the pixels in the binary mask which are not part of the k tracks

obtained so far. Our algorithm stops when all pixels in the binary mask have been included in a track.

C.3.2 Track growth

Given an initial seed location, we obtain a new track $\mathbf{S}$ by iteratively growing two single-path tracks:

$$\mathbf{S} \equiv [\mathbf{S}^{(b)}, \mathbf{S}^{(f)}],$$

such that the last segment of $\mathbf{S}^{(b)}$ is connected to the first segment of $\mathbf{S}^{(f)}$. Starting in the middle of a branch means that there will be two possible path directions.

Given a current segment S_t, we determine the center location of the new segment S_{t+1} by minimizing the orientation-weighted SSD between S_t and a set of candidate locations:

$$\mathbf{c}_{t+1} = \underset{\mathbf{c}'}{\operatorname{argmin}} \operatorname{SSD}\left(S_t(\mathbf{c}), S'_d(\mathbf{c}')\right), \tag{C.7}$$

such that $\rho_{\mathbf{c}} \geqslant \|\mathbf{c}-\mathbf{c}'\|_2 \leqslant 2\rho_{\mathbf{c}}$ and $\mathbf{c} \notin \mathcal{S}_{t-1}$. Here, $\mathbf{c}'$ is the center location of segment S' and $\operatorname{SSD}(S_t, S'_d)$ is the SSD between two segments aligned at their centers. S'_d is the candidate segment multiplied by the von Mises distribution described above:

$$S'_d(\mathbf{p}) = S'(\mathbf{p})\mathrm{VM}(\mathbf{p}-\mathbf{c}; \delta\mathbf{c}, \kappa),$$

where $\delta\mathbf{c}$ is the current direction of growth.[1] Note that we minimize only with respect to the location of the segment, not with respect to its radii. This constraints the optimization to a two-dimensional search space. The search area over which to minimize (C.7) defines the range of possible step sizes between segments. Since successive segments cannot be disjoint, the new segment center cannot lie beyond $2\rho_{\mathbf{c}}$ of the current center. On the other hand, to discourage very small steps, we set a minimum step size of $\rho_{\mathbf{c}}$. The search space is thus defined over a ring around

[1] The weighting function VM is not used for $t = 0$.

the current segment in which both large and small steps are disallowed. Finally, the second condition prohibits selecting a segment S_{t+1} which overlaps with the prior segment S_{t-1}, as noted in (C.2).

We stop growing a track when the all candidate segment centers lie outside the binary image mask or have already been visited by a previous track. Thus, a single-path track extends from a seed location to either another track or to the border of an area in which there is no pixel data that corresponds to the tree.

Finally, to obtain a full bidirectional track, we first determine a single-path track $\mathbf{S}^{(f)}$ as described above. We then start again from the seed location $S_1^{(f)}$ and obtain a second track $\mathbf{S}^{(b)}$. In the first step of the second run, we ignore the pixels that correspond to the second segment of $S^{(f)}$. This forces the second track to select a path in the opposite direction. We then merge the two tracks into the final track:

$$\mathbf{S} = [\mathbf{S}^{(b)}, \mathbf{S}^{(f)}].$$

C.3.3 Multiple track estimation

To construct a set $\mathfrak{S}$ of K tracks:

$$\mathfrak{S} = \{\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_K\},$$

we iteratively apply the single track estimation algorithm defined above. That is, we first grow the bidirectional track $\mathbf{S}_1$ starting at a pixel with the highest LoG-Gabor response in the image. Then, after growing $\mathbf{S}_1$, we select a highest response pixel that lies outside this track to obtain the second track and so on.

To cover the entire tree, we simply keep growing new tracks until all pixels in the binary mask have been included in a track.

C.4 Planar graph estimation

We convert a set of K tracks that span the binary mask of a tree into a planar graph $G = (V, E)$ as follows. Let C_i be the set of segment centers for the i-th track. Then:

$$V = C_1 \cup C_2 \cup \ldots C_K.$$

Then, we define an edge between all pairs of segments that overlap:

$$(u, v) \in E \Leftrightarrow S_u \cup S_v \neq \varnothing,$$

where S_v is the segment corresponding to vertex v.

Bibliography

Acharya, T. and Ray, A. K. (2005), *Image processing: principles and applications*, Wiley-Interscience.

Aono, M. and Kunii, T. (May), "Botanical Tree Image Generation," *Computer Graphics and Applications, IEEE*, 4, 10–34.

Austrin, P., Khot, S., and Safra, M. (2009), "Inapproximability of vertex cover and independent set in bounded degree graphs," in *Computational Complexity, 2009. CCC'09. 24th Annual IEEE Conference on*, pp. 74–80, IEEE.

Band, L. R., Fozard, J. A., Godin, C., Jensen, O. E., Pridmore, T., Bennett, M. J., and King, J. R. (2012), "Multiscale Systems Analysis of Root Growth and Development: Modeling Beyond the Network and Cellular Scales," *The Plant Cell Online*, 24, 3892–3906.

Beers, M. H., Fletcher, A. J., Jones, T., and Porter, R. (2004), *The Merck manual of medical information*, Pocket Books.

Bejan, A. and Lorente, S. (2010), "The constructal law of design and evolution in nature," *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365, 1335–1347.

Bellman, R. (2003), *Dynamic programming*, Dover.

Benmansour, F. and Cohen, L. (2011), "Tubular structure segmentation based on minimal path method and anisotropic enhancement," *International Journal of Computer Vision*, 92, 192–210.

Bishop, C. (2006), *Pattern recognition and machine learning*, vol. 4, Springer New York.

Buchin, K. and Schulz, A. (2010), "On the Number of Spanning Trees a Planar Graph Can Have," in *Algorithms – ESA 2010*, eds. M. Berg and U. Meyer, vol. 6346 of *Lecture Notes in Computer Science*, pp. 110–121, Springer Berlin Heidelberg.

Chaudhuri, S., Chatterjee, S., Katz, N., Nelson, M., and Goldbaum, M. (1989), "Detection of blood vessels in retinal images using two-dimensional matched filters," *IEEE Transactions on Medical Imaging*, 8, 263–269.

Chen, X., Neubert, B., Xu, Y.-Q., Deussen, O., and Kang, S. B. (2008), "Sketch-based tree modeling using Markov random field," *ACM Trans. Graph.*, 27, 109:1–109:9.

Cheng, H. D., Jiang, X. H., Sun, Y., and Wang, J. L. (2001), "Color image segmentation: Advances and prospects," *Pattern Recognition*, 34, 2259–2281.

Chimani, M., Kandyba, M., Ljubić, I., and Mutzel, P. (2010), "Obtaining optimal k-cardinality trees fast," *J. Exp. Algorithmics*, 14, 5:2.5–5:2.23.

Coelho, M., Villalobos, F., and Mateos, L. (2003), "Modeling root growth and the soil–plant–atmosphere continuum of cotton crops," *Agricultural water management*, 60, 99–118.

Cormen, T., Leiserson, C., Rivest, R., and Stein, C. (2001), *Introduction to algorithms*, MIT Press.

Das, A., Lauffenburger, D., Asada, H., and Kamm, R. D. (2010), "A hybrid continuum–discrete modelling approach to predict and control angiogenesis: analysis of combinatorial growth factor and matrix effects on vessel-sprouting morphology," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 368, 2937–2960.

Dashtbozorg, B., Mendonca, A., and Campilho, A. (2013), "An Automatic Graph-based Approach for Artery/Vein Classification in Retinal Images," *Image Processing, IEEE Transactions on*, pp. 1–1.

Di Battista, G. and Tamassia, R. (1996), "On-Line Planarity Testing," *SIAM Journal on Computing*, 25, 956–997.

Diestel, R. (2010), *Graph theory. 2010*, Springer, 4 edn.

Dijkstra, E. (1959), "A note on two problems in connexion with graphs," *Numerische Mathematik*, 1, 269–271.

Dupuy, L., Gregory, P. J., and Bengough, A. G. (2010), "Root growth models: towards a new generation of continuous approaches," *Journal of experimental botany*, 61, 2131–2143.

Estrada, R., Tomasi, C., Trager, M., Wallace, D., Freedman, S., and Farsiu, S. (2011), "Enhanced video indirect ophthalmoscopy (VIO) via robust mosaicing," *Biomedical Optics Express*, 2, 2871–2887.

Estrada, R., Tomasi, C., Cabrera, M. T., Wallace, D. K., Freedman, S. F., and Farsiu, S. (2012), "Exploratory Dijkstra forest based automatic vessel segmentation: applications in video indirect ophthalmoscopy (VIO)," *Biomedical optics express*, 3, 327–339.

Fortin, S. (1996), "The graph isomorphism problem," Tech. rep., Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada.

Friman, O., Hindennach, M., Kuhnel, C., and Peitgen, H. (2010), "Multiple hypothesis template tracking of small 3D vessel structures," *Medical image analysis*, 14, 160–171.

Galkovskyi, T., Mileyko, Y., Bucksch, A., Moore, B., Symonova, O., Price, C., Topp, C., Iyer-Pascuzzi, A., Zurek, P., Fang, S., et al. (2012), "GiA Roots: software for the high throughput analysis of plant root system architecture," *BMC Plant Biology*, 12, 116.

Gao, X., Xiao, B., Tao, D., and Li, X. (2010), "A survey of graph edit distance," *Pattern Analysis and Applications*, 13, 113–129.

Garey, M. R. and Johnson, D. S. (1979), *Computers and intractability*, vol. 174, Freeman San Francisco, CA.

Gonzalez, R. C. and Woods, R. E. (2002), *Digital image processing*, Prentice Hall New Jersey.

Gou, X., Chen, M., Zhang, Y., Dong, W., and Qie, X. (2009), "Wavelet multiresolution based multifractal analysis of electric fields by lightning return strokes," *Atmospheric Research*, 91, 410–415.

Graham, M., Gibbs, J., and Higgins, W. (2008), "Robust system for human airway-tree segmentation," in *Proceedings of SPIE*, vol. 6914, p. 69141J.

Gschwantner, T., Schadauer, K., Vidal, C., Lanz, A., Tomppo, E., di Cosmo, L., Robert, N., Englert Duursma, D., and Lawrence, M. (2009), "Common tree definitions for national forest inventories in Europe," *Silva Fennica*, 43, 303–321.

Gülsün, M. and Tek, H. (2008), "Robust Vessel Tree Modeling," *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2008*, pp. 602–611.

Hendargo, H. C., Estrada, R., Chiu, S. J., Tomasi, C., Farsiu, S., and Izatt, J. A. (2013), "Automated non-rigid registration and mosaicing for robust imaging of distinct retinal capillary beds using speckle variance optical coherence tomography," *Biomed. Opt. Express*, 4, 803–821.

Hildebrand, G. D. and Fielder, A. R. (2011), "Anatomy and Physiology of the Retina," in *Pediatric Retina*, eds. J. Reynolds and S. Olitsky, pp. 39–65, Springer Berlin Heidelberg.

Hoover, A., Kouznetsova, V., and Goldbaum, M. (2002), "Locating blood vessels in retinal images by piecewise threshold probing of a matched filter response," *IEEE Transactions on Medical Imaging*, 19, 203–210.

Ijiri, T., Owada, S., and Igarashi, T. (2006), "The Sketch L-System: Global Control of Tree Modeling Using Free-Form Strokes," in *Smart Graphics*, eds. A. Butz, B. Fisher, A. Krüger, and P. Olivier, vol. 4073 of *Lecture Notes in Computer Science*, pp. 138–146, Springer Berlin Heidelberg.

Iyer-Pascuzzi, A., Zurek, P., and Benfey, P. (2013), "High-Throughput, Noninvasive Imaging of Root Systems," in *Plant Organogenesis*, ed. I. De Smet, vol. 959 of *Methods in Molecular Biology*, pp. 177–187, Humana Press.

Kandel, E. R., Schwartz, J. H., and Jessell, T. M. (2000), *Principles of neural science*, vol. 4, McGraw-Hill New York.

Kapoor, S. and Ramesh, H. (2000), "An Algorithm for Enumerating All Spanning Trees of a Directed Graph," *Algorithmica*, 27, 120–130.

Khatri, C. and Mardia, K. (1977), "The von Mises-Fisher matrix distribution in orientation statistics," *Journal of the Royal Statistical Society. Series B (Methodological)*, 39, 95–106.

Khuller, S., Raghavachari, B., and Young, N. (1995), "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, 14, 305–321.

Kirbas, C. and Quek, F. (2004), "A review of vessel extraction techniques and algorithms," *ACM Computing Surveys*, 36, 81–121.

Koene, R., Tijms, B., Hees, P., Postma, F., Ridder, A., Ramakers, G., Pelt, J., and Ooyen, A. (2009), "NETMORPH: A Framework for the Stochastic Generation of Large Scale Neuronal Networks With Realistic Neuron Morphologies," *Neuroinformatics*, 7, 195–210.

Kuhn, H. W. (1955), "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, 2, 83–97.

Lambert, D. (1992), "Zero-inflated Poisson regression, with an application to defects in manufacturing," *Technometrics*, 34, 1–14.

Larkman, A. U. (1991), "Dendritic morphology of pyramidal neurons of the visual cortex of the rat: I. Branching patterns," *The Journal of Comparative Neurology*, 306, 307–319.

Larrosa, J. and Schiex, T. (2004), "Solving weighted {CSP} by maintaining arc consistency," *Artificial Intelligence*, 159, 1 – 26.

Li, H. and Yezzi, A. (2007), "Vessels as 4-D curves: Global minimal 4-D paths to extract 3-D tubular surfaces and centerlines," *IEEE Transactions on Medical Imaging*, 26, 1213–1223.

Li, Q., You, J., Zhang, L., and Bhattacharya, P. (2006), "Automated retinal vessel segmentation using Gabor filters and scale multiplication," *SMC*, 6, 3521–3527.

Lo, P., Sporring, J., Pedersen, J., and de Bruijne, M. (2009), "Airway tree extraction with locally optimal paths," *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2009*, pp. 51–58.

Lo, P., Sporring, J., Ashraf, H., Pedersen, J. J., and de Bruijne, M. (2010), "Vessel-guided airway tree segmentation: A voxel classification approach," *Medical Image Analysis*, 14, 527 – 538.

López, L. D., Ding, Y., and Yu, J. (2010), "Modeling Complex Unfoliaged Trees from a Sparse Set of Images," *Computer Graphics Forum*, 29, 2075–2082.

López-Cruz, P. L., Bielza, C., Larrañaga, P., Benavides-Piccione, R., and DeFelipe, J. (2011), "Models and Simulation of 3D Neuronal Dendritic Trees Using Bayesian Networks," *Neuroinformatics*, 9, 347–369.

Lynn, B. H., Yair, Y., Price, C., Kelman, G., and Clark, A. J. (2012), "Predicting cloud-to-ground and intracloud lightning in weather forecast models," *Weather and Forecasting*, 27, 1470–1488.

Macklin, P., McDougall, S., Anderson, A. R., Chaplain, M. A., Cristini, V., and Lowengrub, J. (2009), "Multiscale modelling and nonlinear simulation of vascular tumour growth," *Journal of mathematical biology*, 58, 765–798.

Mansell, E. R., MacGorman, D. R., Ziegler, C. L., and Straka, J. M. (2002), "Simulated three-dimensional branched lightning in a numerical thunderstorm model," *Journal of Geophysical Research: Atmospheres*, 107, ACL 2–1–ACL 2–12.

Movellan, J. (2002), "Tutorial on gabor filters," *Open Source Document*.

Osareh, A., Mirmehdi, M., Thomas, B., and Markham, R. (2003), "Automated identification of diabetic retinal exudates in digital colour images," *British Journal of Ophthalmology*, 87, 1220–1223.

Pechaud, M., Keriven, R., and Peyre, G. (2009), "Extraction of tubular structures over an orientation domain," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 336–342, IEEE.

Peirce, S. M. (2010), "Computational and mathematical modeling of angiogenesis," *Microcirculation*, 15, 739–751.

Peli, E. (1990), "Contrast in complex images," *Journal of the Optical Society of America A*, 7, 2032–2040.

Perez, M., Highes, A., Stanton, A., Thorn, S., Chapman, N., Bharath, A., and Parker, K. (2002), "Retinal vascular tree morphology: a semi-automatic quantification," *Biomedical Engineering, IEEE Transactions on*, 49, 912–917.

Perfahl, H., Byrne, H. M., Chen, T., Estrella, V., Alarcón, T., Lapin, A., Gatenby, R. A., Gillies, R. J., Lloyd, M. C., Maini, P. K., et al. (2011), "Multiscale modelling of vascular tumour growth in 3D: the roles of domain size and boundary conditions," *PloS one*, 6, e14790.

Prusinkiewicz, P. and Lindenmayer, A. (1991), *The algorithmic beauty of plants (The Virtual Laboratory)*, Springer.

Quinas-Guerra, M., Ribeiro-Rodrigues, T., Rodríguez-Manzaneque, J. C., and Travasso, R. D. (2012), "Understanding the Dynamics of Tumor Angiogenesis: A Systems Biology Approach," *Systems Biology in Cancer Research and Drug Discovery*, pp. 197–227.

Rakov, V. V. A. and Uman, M. A. (2003), *Lightning: physics and effects*, Cambridge University Press.

Rattathanapad, S., Mittrapiyanuruk, P., Kaewtrakulpong, P., Uyyanonvara, B., and Sinthanayothin, C. (2012), "Vessel extraction in retinal images using multilevel line detection," in *Biomedical and Health Informatics (BHI), 2012 IEEE-EMBS International Conference on*, pp. 345–349.

Raven, J. A. and Edwards, D. (2001), "Roots: evolutionary origins and biogeochemical significance," *Journal of Experimental Botany*, 52, 381–401.

Reche-Martinez, A., Martin, I., and Drettakis, G. (2004), "Volumetric reconstruction and interactive rendering of trees from photographs," *ACM Trans. Graph.*, 23, 720–727.

Reffye, P., Kang, M., Hua, J., and Auclair, D. (2012), "Stochastic modelling of tree annual shoot dynamics," *Annals of Forest Science*, 69, 153–165.

Ricci, E. and Perfetti, R. (2007), "Retinal blood vessel segmentation using line operators and support vector classification," *IEEE Transactions on Medical Imaging*, 26, 1357–1365.

Rogers, L. C. G. and Williams, D. (2000), *Diffusions, Markov processes and martingales: Volume 2, Itô calculus*, vol. 2, Cambridge university press.

Rothaus, K., Rhiem, P., and Jiang, X. (2007), "Separation of the Retinal Vascular Graph in Arteries and Veins," in *Graph-Based Representations in Pattern Recognition*, eds. F. Escolano and M. Vento, vol. 4538 of *Lecture Notes in Computer Science*, pp. 251–262, Springer Berlin Heidelberg.

Schaap, M., Smal, I., Metz, C., van Walsum, T., and Niessen, W. (2007), "Bayesian tracking of elongated structures in 3D images," in *Information Processing in Medical Imaging*, pp. 74–85, Springer.

Soares, J., Leandro, J., Cesar Jr, R., Jelinek, H., and Cree, M. (2006), "Retinal vessel segmentation using the 2-D Gabor wavelet and supervised classification," *IEEE Transactions on Medical Imaging*, 25, 1214–1222.

Starck, J.-L., Murtagh, F., Candes, E. J., and Donoho, D. L. (2003), "Gray and color image contrast enhancement by the curvelet transform," *Image Processing, IEEE Transactions on*, 12, 706–717.

Tan, P., Fang, T., Xiao, J., Zhao, P., and Quan, L. (2008), "Single image tree modeling," *ACM Trans. Graph.*, 27, 108:1–108:7.

Tarjan, R. (1976), "Edge-disjoint spanning trees and depth-first search," *Acta Informatica*, 6, 171–185.

Toroslu, I. H. and Üçoluk, G. (2007), "Incremental assignment problem," *Information Sciences*, 177, 1523 – 1529.

Türetken, E., Blum, C., González, G., and Fua, P. (2010), "Reconstructing geometrically consistent tree structures from noisy images," in *Proceedings of the 13th international conference on Medical image computing and computer-assisted intervention: Part I*, MICCAI'10, pp. 291–299, Berlin, Heidelberg, Springer-Verlag.

Türetken, E., González, G., Blum, C., and Fua, P. (2011), "Automated Reconstruction of Dendritic and Axonal Trees by Global Optimization with Geometric Priors," *Neuroinformatics*, 9, 279–302.

Vos, J., Evers, J. B., Buck-Sorlin, G. H., Andrieu, B., Chelle, M., and de Visser, P. H. B. (2010), "Functional-structural plant modelling: a new versatile tool in crop science," *Journal of Experimental Botany*, 61, 2101–2115.

Watson, H. and Galton, F. (1875), "On the probability of the extinction of families," *The Journal of the Anthropological Institute of Great Britain and Ireland*, 4, 138–144.

Willard, S. (2004), *General topology*, Dover Publications.

Wilson, C., Cocker, K., Moseley, M., Paterson, C., Clay, S., Schulenburg, W., Mills, M., Ells, A., Parker, K., Quinn, G., et al. (2008), "Computerized analysis of retinal vessel width and tortuosity in premature infants," *Investigative Ophthalmology & Visual Science*, 49, 3577–3585.

Wink, O., Niessen, W., and Viergever, M. (2004), "Multiscale vessel tracking," *IEEE Transactions on Medical Imaging*, 23, 130–133.

Xiao, C., Staring, M., Wang, Y., Shamonin, D., and Stoel, B. (2013), "Multiscale Bi-Gaussian Filter for Adjacent Curvilinear Structures Detection With Application to Vasculature Images," *Image Processing, IEEE Transactions on*, 22, 174–188.

Yedidya, T. and Hartley, R. (2008), "Tracking of Blood Vessels in Retinal Images Using Kalman Filter," in *Digital Image Computing: Techniques and Applications (DICTA), 2008*, pp. 52–58.

Zeng, J., Zhang, Y., and Zhan, S. (2006), "3D Tree Models Reconstruction from a Single Image," in *Intelligent Systems Design and Applications, 2006. ISDA '06. Sixth International Conference on*, vol. 2, pp. 445–450.

Zeng, Z., Tung, A. K. H., Wang, J., Feng, J., and Zhou, L. (2009), "Comparing stars: on approximating graph edit distance," *Proc. VLDB Endow.*, 2, 25–36.

Zheng, Y., Gu, S., Edelsbrunner, H., Tomasi, C., and Benfey, P. (2011), "Detailed reconstruction of 3D plant root shape," in *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pp. 2026–2033, Washington, DC, USA, IEEE Computer Society.

Biography

Rolando Estrada is a Ph.D candidate in computer science at Duke University under the guidance of Prof. Carlo Tomasi. Rolando obtained a M.S. in biomedical engineering from Duke University in May 2013 and will also receive a graduate certificate in cognitive neuroscience from Duke in August 2013. He previously obtained a B.A. (summa cum laude) in English in May 2006 and a B.S. (magna cum laude) in computer science in May 2005, both from Louisiana State University, in Baton Rouge, LA.

His research interests focus on the mathematical aspects of artificial intelligence and cognition, such as machine learning, Bayesian analysis, and graph theory, particularly as they apply to the problem of vision. He specializes in computer vision and has had the fortune of applying his expertise to a number of medical imaging applications, including image mosaicing, retinal vessel segmentation and vessel topology estimation.